Proceedings of Science and Mathematics
Faculty of Science, Universiti Teknologi Malaysia
Vol. 24, 2024, page 126-132

The Application of Automatic Derivative in System of Nonlinear Equations

Nur Afiqah Masyitah Mohd Zaki, Yeak Su Hoe
Department of Mathematical Sciences, Faculty of Science, Universiti Teknologi Malaysia
Corresponding author: s.h.yeak@utm.my

Abstract

Automatic differentiation is a useful technique for computing the derivatives of functions that expressed as computer programmes. The accuracy and effectiveness of the derivatives make it indispensable in numerous disciplines, such as machine learning, optimisation, and scientific computing. The purpose of this study is to investigate the applications of automated differentiation (AD) in machine learning and mathematics. Its principal objectives include the investigation of related problems and applications, methods for improving differentiation, and mathematical applications of AD in non-linear systems. PyCharm, a Python programming tool, will be utilised in this study to make the application of mathematics easier. The Jacobian matrix of functions with inputs and outputs is created in AD systems using forward and reverse mode differentiation, it also a computational method for calculating gradients in deep learning issues involving several input variables. When there are numerous input variables, forward mode differentiation typically becomes challenging, however reverse mode requires more data and can compute multiple derivatives at once in a single run. The application of automatic differentiation to parameterized nonlinear system solutions is discussed. It discusses iterations, particularly Newton's approach for parameter optimisation and fixed-point iterations. A comparison is presented between the fixed-point iteration method and Newton's methodology. One particular application of fixed-point iteration is the Newton's method. In general, AD enables the computation of gradients for parameter optimisation and sensitivity analysis in a range of applications.

Keywords: Automatic Derivative; Nonlinear equation; Fixed-Point iteration; Newton’s method

1. Introduction

Calculating derivatives is necessary for many numerical techniques.The gradients are essential to machine learning because they help train neural networks [1]. Computing derivatives becomes a challenging undertaking as probabilistic models and algorithms become more complicated. It can only quantitatively analyse derivatives at specific places in many issues and unable to calculate them analytically. It can be difficult and time-consuming to solve an analytical problem by hand, even in cases where one exists.This paradigm has grown in popularity in recent years because it offers a powerful tool for optimising issues, machine learning, and scientific modelling [2].

The computation of derivatives in computer programs can be categorized into four main methods: manual derivation and coding, numerical differentiation using finite difference approximations, symbolic differentiation and automatic differentiation, also known as algorithmic differentiation. Traditional machine learning methods often rely on the evaluation of derivatives and the introduction of new models has historically involved manually deriving analytical derivatives for optimization procedures. Manual differentiation more error-prone, while numerical differentiation, though simple, can be highly inaccurate and does not scale well for machine learning applications. Symbolic differentiation addresses some issues but can result in complex expressions and limits algorithmic control flow. AD is a powerful technique that interprets a computer program by incorporating derivative values into the variable domain and redefining operators to propagate derivatives according to the chain rule of differential calculus. With the rise of deep learning and modern workflows based on rapid prototyping and code reuse in frameworks like Torch, and TensorFlow, projects such as autogradand PyTorch have played a significant role in bringing general-purpose AD to the mainstream.

This research aims to (1) explain the concept and techniques of automatic differentiation, (2) explore the efficiency and accuracy of AD in non-linear system and (3) investigate how AD is implemented mathematically. This study focused on implementation of AD in non-linear system. Compile time is not taken into account because the purpose only wanted to builds a program application of AD.

2. Literature Review

2.1 The development of AD

AD fundamental mathematical concepts have existed for a very long period. No promise is made here about completeness, as the methodologies of AD have been independently discovered numerous times by different people at different times and places. The indirect approach of first obtaining derivative formulas and then evaluating those appears to have been utilised fairly universally, even though the AD methodology could have been used for the assessment of derivatives by hand or with tables and desk calculators. As such, the topic of debate will be limited to the era of digital computers. The history of AD from 1962 to the present can be broadly categorised into four decades. In the first, a variety of issues were tackled using the straightforward, uncomplicated method known as the forward mode, primarily at the Mathematics Research Centre (MRC) of the University of Wisconsin–Madison [3]. A period of inaction ensued, during which AD was not accepted for unclear reasons. But by 1982, because to advancements in programming methods and the discovery of the effective reverse mode, interest in AD had unquestionably returned. The work of Andreas Griewank [4] and his colleagues at Argonne National Laboratory (ANL) has been crucial in much of the advances made in this age. Subsequently, there was a remarkable surge in research and development of AD approaches, tools, and applications, as documented in the Griewank's book[4]. The computational graph, a method of visualising an algorithm or computer programme other than mathematics, is a helpful tool in the development of AD after 1980.

2.2 *Automatic differentiation with symbolic and numerical differentiation*

Symbolic differentiation is the process of computing the derivative of a function using symbolic expressions [4]**.** While symbolic differentiation is beneficial for basic functions, it can be problematic for complex functions and may suffer from expressions well, where the derivative expression becomes exponentially larger than the original expression. This problem can be overcome by simplifying the derivative, but the process can be costly. Numerical differentiation, on the other hand, computes the difference quotient using finite differences to approximate the derivative of a function. This method is straightforward to construct, but it is at risk of numerical inaccuracies and may necessitate a large number of function evaluations, which can be computationally expensive [5].

Automatic differentiation, overcomes the limitations of both symbolic and numerical differentiation by computing derivatives via a recursive application of the chain rule [6]. This method is fast and precise, and it can compute exact derivatives to machine precision for any differentiable function, regardless of its complexity.

2.3 *Automatic derivative type*

Automatic differentiation can be performed in forward, backward, or mixed mode using both the operator overloading approach and the source code transformation method [7]. The variation to use is determined by the situation's unique needs, such as the complexity of the function to be differentiated, the precision and efficiency of the differentiation, and the programming language and tools available. Operator overloading, for example, may be desirable for tiny functions and rapid prototyping, but source code translation may be preferable for complicated functions and production-level code.

2.4 *Gradient Based optimisation*

Gradient-based optimization involves using gradient information to guide the search for optimal

solutions. *Principles and Techniques of Algorithmic Differentiation*, systematically covers the theoretical foundations of AD and its application in optimization, emphasizing its superiority over numerical differentiation in terms of accuracy and efficiency [4].

2.5 *The use of python as mathematical programming*

Automatic differentiation automatically differentiates numerical Python code, reducing simulation times by less than 4 times compared to hand-written differentiation code [8].

3. Methodology

3.1. Research Data

AD enables the computation of precise derivatives in a specific amount of time. The applications of this sensitivity analysis are numerous. Differentiation is used in the backpropagation technique in deep neural networks and in any other field where a rate of change needs to be defined. A derivative is a way to quantify rate changes. The fastest optimisation methods rely on derivative computation. Assuming, for instance, that the square root function is the inverse of the square function, the inverse of the function f, which involves many times, can be calculated using Newton's approach as indicated in equation (1).

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{1}$$

where x_n represents the value of a variable at a specific iteration or step, and n represents the index or number of that iteration. The process continues until convergence, then the derivatives are used.

1.1. Automatic differentiation (AD) stages

The stages in the analysis of AD using following technique and approach:

1) Dual number.
2) Forward mode differentiation
3) Reverse mode differentiation (backpropagation)

Analyzing AD using the following computational graf function in equation (2):
Determine the following functions using forward and reverse mode:

$$(x, y) = xy - \sin(y) \tag{2}$$

By using forward mode and backward mode, to calculate $\frac{\partial f}{\partial y}$ defined in equation (3) and (4):

$$\dot{v}_i = \frac{\partial v_i}{\partial y} \tag{3}$$

$$\bar{v}_i = \frac{\partial f_j}{\partial v_i} \tag{4}$$

4. Results and discussion

4.1. Newton's method

Newton's method is a faster approach for finding the convergence. It also mentions the use of approximate Jacobian updates in practical implementations to improve computational efficiency [9]. By using function in nonlinear system as in equation (5):

$$\begin{aligned} 2x + y + xy - 1 &= 0 \\ x + 2y + x^2 - 1 &= 0 \end{aligned} \tag{5}$$

the multidimensional Taylor series in (6) can be simplified as (7):

$$A_1(a,b) + [B_{11}B_{12}]\begin{bmatrix} x \\ y \end{bmatrix}$$

$$A_2(a,b) + [B_{21}B_{22}]\begin{bmatrix} x \\ y \end{bmatrix} \tag{6}$$

$$F = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} A_1(a,b) \\ A_2(a,b) \end{bmatrix} + [B(a,b)\begin{bmatrix} x \\ y \end{bmatrix} \tag{7}$$

Using newton iteration, it can be form as in equation (8):

$$X_{n+1} = X_n - J(X_n)^{-1} \cdot F(X_n) \ . \tag{8}$$

Using property, the whole iteration form can derived as equation (9),

$$X_{n+1} = G_n X_n + H_n. \tag{9}$$

where$W_n = G_n \cdots G_0,\ H_n = G_n \cdots G_1 K_0 + G_n \cdots G_2 K_1 + G_n \cdots G_3 K_2 + \cdots + G_n K_{n-1} + K_n$

with the convergence $||G_0|| < 1$.

4.2 Convergence result newton's method

Using python programming, with feasible initial guess of X_0, Figure 4.2 shows the result of Newton's method in nonlinear system with two variable. It shows that result converged after 5 iterations.

```
Fxad=  tensor([-0.5429, -2.2500], grad_fn=<StackBackward0>)
jacobian=  tensor([[-0.0099,  0.5000],
        [ 1.0000,  1.0000]])
Iteration: 0
Current xad: tensor([0.8000, 0.8000], requires_grad=True)
F(xad): tensor([2.1811, 2.0400], grad_fn=<StackBackward0>)
Jacobian: tensor([[2.0068, 1.8000],
        [1.0000, 3.6000]])
Update step: tensor([-0.7706, -0.3526], grad_fn=<NegBackward0>)
Updated xad: tensor([0.0294, 0.4474], requires_grad=True)
Iteration: 1
Current xad: tensor([0.0294, 0.4474], requires_grad=True)
F(xad): tensor([0.4812, 0.1243], grad_fn=<StackBackward0>)
Jacobian: tensor([[1.1548, 1.0294],
        [1.0000, 2.8948]])
Update step: tensor([-0.5467,  0.1459], grad_fn=<NegBackward0>)
Updated xad: tensor([-0.5173,  0.5933], requires_grad=True)
Iteration: 2
Current xad: tensor([-0.5173,  0.5933], requires_grad=True)
F(xad): tensor([-0.0149,  0.0213], grad_fn=<StackBackward0>)
Jacobian: tensor([[1.0776, 0.4827],
        [1.0000, 3.1866]])
```

```
Update step: tensor([ 0.0196, -0.0128], grad_fn=<NegBackward0>)
Updated xad: tensor([-0.4977,  0.5805], requires_grad=True)
Iteration: 3
Current xad: tensor([-0.4977,  0.5805], requires_grad=True)
F(xad): tensor([-0.0002,  0.0002], grad_fn=<StackBackward0>)
Jacobian: tensor([[1.0714, 0.5023],
        [1.0000, 3.1609]])
Update step: tensor([ 0.0002, -0.0001], grad_fn=<NegBackward0>)
Updated xad: tensor([-0.4975,  0.5803], requires_grad=True)
Iteration: 4
Current xad: tensor([-0.4975,  0.5803], requires_grad=True)
F(xad): tensor([0., 0.], grad_fn=<StackBackward0>)
Converged after 5 iterations.
Converged xad: tensor([-0.4975,  0.5803], requires_grad=True)
[-0.49745408  0.58033353]

Process finished with exit code 0
```

Figure 4.1 Result of newton's method in nonlinear system with two variables.

4.3 Fixed point method

To stages to analysis using fixed-point method

1. Input data using equation (5)
2. Derive equation (8) using fixed-point iteration with feasible initial guess, it can be expressed as equation (9) and can be simplified into equation (10)

$$X_{n+1} = G_n X_n \,, \tag{9}$$

$$X_{n+1} = W_n X_0 \,, \tag{10}$$

where $W_n = G_n \cdots G_0$.

A study that emphasis the necessity of regularisation in inverse problem solving and the combination of data-driven regularisation and convex feasibility, supports this study on fixed-point iteration in deep learning [10].

4.3 Debugging python program

The code failed to run after encountering an error during execution. However, this event made it necessary to look into the issue more thoroughly, which helped to clarify the debugging procedure. Important insights on debugging and improving software were obtained by methodically finding and fixing the flaws in the code. Thus, this experience helped to deepen the understanding of error detection and resolution approaches, as well as emphasising the value of debugging in programming. A debugging tool is a type of software used for debugging computer programs which allows stopping code execution at any given point,restarting it and continue stepping through the text as desired [12]). Developers will face limitless frustration trying to solve their problems if they cannot debug code. Developers can consider themselves lucky because they can easily find and correct their programs'

weaknesses without losing much time using Python debugging tactics and integrated solutions. Figure 4.2 shows the Pycharm ‘s debugger.

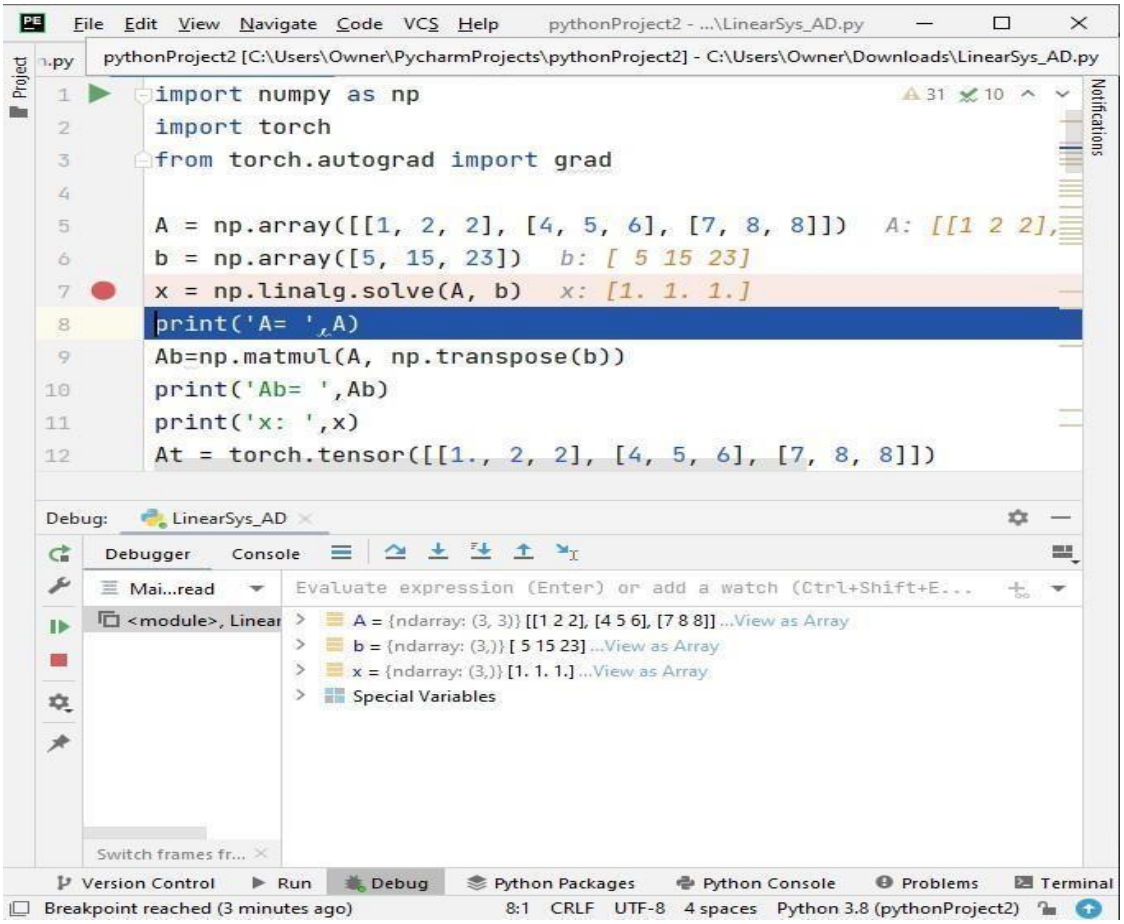

Figure 4.2 the Pycharm ‘s debugger.

Conclusion

In many different domains, the research and use of nonlinear systems have been greatly influenced by Automatic Differentiation (AD). AD makes use of mathematical ideas like the dual numbers and Taylor's theorem to facilitate accurate and speedy derivative computing, which is essential for the analysis and optimisation of nonlinear systems. AD guarantees precise gradient computation in the setting of nonlinear optimisation, which is necessary for gradient-based optimisation techniques. This accuracy improves optimisation methods, leading to more dependable and rapid convergence. Large-scale, complex nonlinear optimisation issues that arise in machine learning can be handled using it. The ability of AD to compute derivatives efficiently also makes it easier to solve nonlinear equations and systems, which are frequently needed in optimisation assignments. Reverse mode AD is used by the backpropagation algorithm, which is the foundation of deep learning, to effectively compute accurate gradients. The first-order Taylor expansion is utilised by forward mode AD with dual numbers to enable accurate and efficient gradient computation for nonlinear functions. This method works especially well for issues where there aren't as many input variables, making nonlinear system optimisations quicker and more accurate. Because Backward mode AD uses the chain rule to effectively compute gradients for functions with multiple inputs, it is the best option for training large nonlinear systems like deep neural networks. The crucial role of reverse mode differentiation is emphasised in [11],especially in deep learning problems where the number of input variables is significantly more than the number of outputs.

Acknowledgement

The researcher would like to thank all people who have supported this research, especially Dr Yeak Su Hoe for his dedication and guidance supervision. Also, appreaciation to friend who have encourage me throughout this process and never forgetting my parents for their cooperation. I am deeply grateful for not giving up on myself which have allowed me to complete my project report.

References

[1] Widrow, B., & Lehr, M. A. (1990). 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. Proceedings of the IEEE, 78(9), 1415-1442.

[2] Mike Innes, Alan Edelman, KenoFischer, Chris Rackauckas, Elliot Saba, Viral B. Shah ,and Will Tebbutt. A Differentiable Programming System to Bridge Machine Learning and Scientific Computing, July2019. arXiv: 1907.07587[cs].

[3] L. B. Rall. Automatic Differentiation: Techniques and Applications, volume 120 of Lecture Notes in Computer Science. Springer, Berlin, 1981.

[4] Andreas Griewank and Andrea Walther. Evaluating derivatives: principles and techniques of algorithmic differentiation. SIAM, 2008..

[5] William H.Press, Saul A.Teukolsky, William T. Vetterling, and Brian P. Flannery.Numerical recipes 3rd edition:The art of scientific computing.Cambridge university press, 2007.

[6] Atilim Gunes Baydin,Barak A. Pearl mutter, Alexey And reyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. Journal of Machine Learning Research, 18:1–43, 2018. Publisher: Microtome Publishing.

[7] Jarrett Revels, Tim Besard, Valentin Churavy, Bjorn DeSutter, and Juan Pablo Vielma. Dynamic automatic differentiation of GPU broadcast kernels. arXiv preprint arXiv:1810.08297,2018.

[8] Nobel, P. (2020). auto_diff: An Automatic Differentiation Package For Python. *2020 Spring Simulation Conference (SpringSim)*, 1-12. https://doi.org/10.22360/SpringSim.2020.ANSS.006.

[9] Campbell, S. L., & Hollenbeck, R. (1996). Automatic differentiation and implicit differential equations. Computational Differentiation: Techniques, Applications, and Tools, 215-227.

[10] Heaton, H., Wu Fung, S., Gibali, A., & Yin, W. (2021). Feasibility-based fixed point networks. Fixed Point Theory and Algorithms for Sciences and Engineering, 2021(1), 1-19.

[11] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M. (2018). Automatic differentiation in machine learning: a survey. Journal of Marchine Learning

[12] Langtangen, H. P. (2014). Debugging in Python. *Center for Biomedical Computing, Simula Research Laboratory, Department of Informatics, University of Oslo*, 1-3.