Numerical Analysis Project1

Gaoshijie .Author

Abstract: This article discusses how the chain rule based on differential function, in addition to achieve the most basic numerical differentiation, or to achieve the differential algorithm, prior to and after the differential algorithm, and analyzes the precision of different algorithms and time-consuming, and draw the conclusion: when the dimension of the output is greater than the input, suitable for use prior to model differential; Reverse mode differentiation is preferred when the output dimension is much smaller than the input. The reason can be seen from the perspective of the number of matrix multiplication, the difference between the forward mode and the reverse mode lies in the difference in the starting point of matrix multiplication. When the output dimension is smaller than the input dimension, the multiplication times of the reverse mode are smaller than that of the forward mode

Keywords: Numerical differentiation,AD forward,AD backward,accuracy,run-time.

1. INTRODUCTION

In this paper, numerical differentiation, forward differentiation and backward differentiation are realized. Numerical differentiation is approximately solved by computing the derivative in the mathematical sense.Due to the problem of computer precision, the exact solution cannot be obtained.

The forward differential and the backward differential are two different algorithms based on the same source, the chain rule for derivatives, and the forward differential is from right to left which is known$\frac{d_w}{d_x}$and want to find out $\frac{d_y}{d_w}$,and the backword derivative is the opposite, from left to right,which is known$\frac{d_y}{d_w}$and want to find out $\frac{d_w}{d_x}$.For forward differential, its solution mode is natural, that is, the mode in accordance with the normal calculation order.

A forward propagation of forward mode can calculate the output value and derivative value. Therefore, for AD forward algorithm ,the project defines an adv class, the class contains a variable and its derivative, and through the list of the initial point of value and its derivative value stored inside, then adv class reloading the all related operator, operator and the actual operator for variable has been, and for its derivative is whether it is a unary operator or a binary operator, both by derivative between derivative algorithm with constant changes. Once defined, you simply define the functions required by the problem.

Backward differentiation is more complicated than forward differentiation, which requires forward propagation to calculate the output value, and then backward propagation to calculate the derivative value, so it has a little more memory overhead, because it needs to save the intermediate variable value in forward propagation, and these variable values are used to calculate the derivative in back propagation.Therefore, for AD backward algorithm,this project chose to record the value of each node, its derivative value and the relationship between it and adjacent nodes as the original function and the derivative function.Therefore, two classes need to be defined, one is cgraph class, which is used to actually calculate the value of each node and its derivative value. The value of each node is no doubt related to ordinary calculation, and the derivative value is calculated by the relationship between each node recorded in node class to calculate the sum value as its derivative value, because of this formula: if y=f(u(x),v(x)),then$\frac{d_y}{d_x}=\frac{d_y}{d_u}*\frac{d_u}{d_x}+\frac{d_y}{d_v}*\frac{d_v}{d_x}$ In the Node class, overloading of a node and its operators is defined, which includes value, derivative value, operator and connection relation. Therefore, this project first calculates the value of the function in a positive way, and then calculates the partial derivative value of each variable in reverse according to the operational relation between each node.

2. CODE DESIGN

Next we see a few subsections.

2.1 Numerical differentiation using finite differences

formula

$$\lim_{h\to 0}\frac{f(x_1,...,x_n+h)-f(x_1,...x_n)}{h}=\frac{\partial f}{\partial x_n}$$

According to the partial derivative formula, we need to take the value of h as small as possible, but considering the existence of numerical error, so the program takes h=10^{-8},then build three lists, one holds f(x),another holds f(x+h),and final one holds the result which is $\frac{f(x+h)-f(x)}{h}$.

2.2 ADforward

formula

$$\frac{\partial f}{\partial x}=(\frac{\partial f}{\partial w_n}(\frac{\partial w_n}{\partial w_{n-1}}...(\frac{\partial w_2}{\partial w_1}\frac{\partial w_1}{\partial x})...)))$$

Forward differentiation is a program that computes partial derivatives from left to right.
First set an adv class contains the value and the derivative value, and the different operator overloading, such as for France, defined function multiplication, because this is a binary operator, so need to two variables, at the same

time it would have two different cases, one is out of the original variables, the other variables are adv, so for value only needs $x_1.a$*$x_2.a$,and for the derivative, we should let value be $x_1.a * x_2.b$+$x_2.b * x_1.a$.If the second variable is a numerical variable, then the value is$x_1.a$*x_2 and the derivative is $x_1.b$*x_2

After the basic class is complete, several computable functions are created.First use sin,cos,ln to optimization function name,The diff function is then defined to return a list of initial variables,the function function defines f(x) as problem requires,the function getdiff is defined to return the list of all the derivative of the incoming list,and finally defines h(x,d) function.This function has two variables for the user to input, one is x, which is a list, representing the initial point, and the other is d, which is a directional square. In this program, in order to solve the Jacobian matrix, the d passed in is the identity matrix.The implemented function h is solvable for x and d of any length,but for the passed function f, if you need to pass in another function, you should change the function function to what you want to explain.

note The submitted program has Jacobian output and time output .

2.3 ADbackforward

$$\frac{\partial f}{\partial x} = (((...(\frac{\partial f}{\partial w_n}\frac{\partial w_n}{\partial w_{n-1}})...\frac{\partial w_2}{\partial w_1})\frac{\partial w_1}{\partial x})$$

Forward differentiation is a program that computes partial derivatives from right to left.
Compared with the forward differential, the backward differential implementation is a lot of trouble, because the backward differential needs to be pre-selected to find the value of the function, and need to save the process of finding the relationship between variables, so as to be able to carry out the reverse calculation.

This program first defines the Node class for calculating graph class CGraph and graph, in which CGraph contains forward evaluation and reverse derivative functions, while Node contains overloaded calculation functions.For each node, it defines its value and its derivative, then defines the OP that records the operation performed, connects the arcs of the derivatives before and after, and overloads the operator so that each operation can store the relation with which node and what operation was performed.

For the Cgraph class, each element is defined as node, and then the main functions computegradient is defined.For the computegradient function, it forms the formula by callback to all nodes and connected nodes, and determines which operation is performed by op stored previously, so as to select the formula of derivation.
Let's take the sine of origin formula as an example,w_4=sin(w_3) ,then firstly node.op is Node.operators.sin will determines that the function originally executed is sin,since it's taking the derivative from left to right, it's going to be $\frac{\partial f}{\partial w_3} = \frac{\partial f}{\partial w_4}\frac{\partial w_4}{\partial w_3} = \frac{\partial f}{\partial w_4}cos(w_3)$,That is, the derivative of the node times the cos(value) of the above operation node.self.The statement executed in the program is"NodeList[node.arg1].derivative +=node.derivative *np. cos(self.NodeList[node.arg1].value)": NodeList[node.arg1] is previous node and node.derivative is current node's derivative.And because a node may perform more than one operation,based on formula $\frac{d_y}{d_x}=\frac{d_y}{d_u}*\frac{d_u}{d_x}+\frac{d_y}{d_v}*\frac{d_v}{d_x}$we consider it in take the cumulative value of multiple computations as the derivative value,that is why the symbol in the program is"+=" not "=".

After defining the class, we define the function function to pass the required function into the program.Then define h(x,d),x is the starting point, d is the direction matrix.In h function ,the first is to import the defined function function and generate the graph,then call the computegradient function to compute the derivative of each node,finally, the derivative is inserted into the matrix and repeated twice to form the Jacobian.And then finally we multiply it by the direction matrix, because they want to find the Jacobian, so d here is the identity matrix. The implemented function h is solvable for x and d of any length,but for the passed function f, if you need to pass in another function, you should change the function function to what you want to explain.

note The submitted program has Jacobian output and time output.

3. USER MANUAL

3.1 Numerical differentiation using finite differences

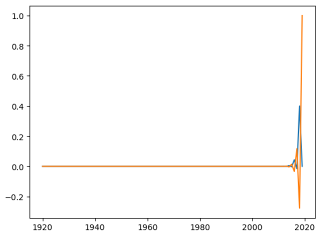
This methods has been realized in Numerical differentiation.py.Running it will print the Jacobian matrix and the running time directly, it doesn't need user to input anything.

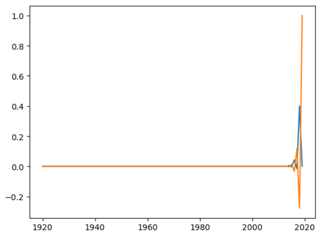
3.2 ADforward

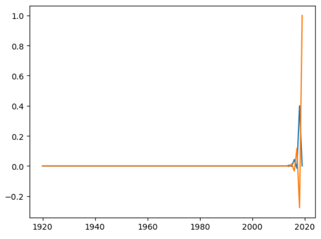
This methods has been realized in ADforward.py.Running it will print the Jacobian matrix and the running time directly, it doesn't need user to input anything.If the user wants to modify the initial point and direction matrix D, he/she can modify it by himself/herself. At present, X and D have been specified in this program according to the title requirements.If you want to modify the passed function, you need to modify the function (x). In this function, if your function needs to be implemented in a loop, as in the problem, and the initial value is constant, then the constant must be initialized to adv(value, 0),otherwise, just return the function is ok.Here is an example,if you want to change the function to x_1^2+x_2^2,you should change the function to:
def function(x):
return [x[0]*x[0]+x[1]*x[1]]
Notice that if you want to add/multiply variable with constant value,the variable must be on the left,which means x[1]*2 is legal but 2*x[1] is not legal.

3.3 ADbackward

This methods has been realized in ADbackward.py.Running it will print the Jacobian matrix and the running time directly, it doesn't need user to input anything.If the user







wants to modify the initial point and direction matrix D, he/she can modify it by himself/herself. At present, X and D have been specified in this program according to the title requirements.If you want to modify the passed function, you need to modify the function (x). In this function, if your function needs to be implemented in a loop, as in the problem, and the initial value is constant, then the constant must be initialized to a=Node(),a.value=constant,otherwise, just return the function is ok.Here is an example,if you want to change the function to $x_1^2+x_2^2$,you should change the function to:
def function(x):
return [x[0]*x[0]+x[1]*x[1]]

4. NUMERICAL RESULTS AND ANALYSIS

Top to bottom are the graphs of Numerical differentiation using finite differences,ADforward,ADbackword.

The horizontal axis is the index of the partial derivative, and the vertical axis is the value of the partial derivative.The blue line is the first row of the Jacobian, and the orange line is the second row of the Jacobian.Since the previous partial derivatives are all zero or almost zero, only the last 100 items are shown

4.1 Analysis

First of all, in terms of time, the time of numerical differentiation is 5.831534385681152s,the time of ADforward is 71.84101033210754,the time of ADforward 0.32512998580932617,and we can see that the ADbackword has the shortest time ,Numerical differentiation follows and ADforward is the longest.We compare ADforward with ADbackward,when the output dimension is larger than the input, it is appropriate to use forward mode differentiation.Reverse mode differentiation is preferred when the output dimension is much smaller than the input.From the point of view of the number of matrix multiplication, the difference between the forward mode and the reverse mode lies in the difference in the starting point of matrix multiplication.When the output dimension is smaller than the input dimension, the multiplication times of the reverse mode are smaller than that of the forward mode.

In terms of accuracy,ADforward and ADbackward are equal and AD can get a more accurate derivative value because we have been calculating the combination of basic elementary functions without making an approximation, and the numerical derivative value has a large deviation from the actual derivative value because h is taken as an approximation and there are machine errors.

5. CONCLUSION

In this paper, the basic numerical differential algorithm, forward differential algorithm and backward differential algorithm are implemented.How can the value of h for numerical differentiation lead to a more precise solution, and the accuracy of forward and backward differentiation is only discussed qualitatively but not quantitatively, which can be further improved.