Optimization Homework 3: Calculating Derivatives

Mark C. Anderson

February 22, 2021

1 Introduction

There are many different methods for calculating derivatives of functions and constraints. These methods vary in accuracy and speed. The goal would be to find a method that is both fast and able to return numerically-exact derivatives. Such a method would be an ideal candidate for use in actual optimization.

This report records the results of three different methods for calculating derivatives: finite differencing, complex step, and algorithmic differentiation. Of these three methods, the complex step method is found to be the fastest, and is known to return numerically-exact derivatives. The complex step method is then successfully used as the method for calculating Jacobians provided to MATLAB's fmincon() function.

Code for this project can be found on GitHub[1] and all methods are discussed in detail in Reference [1].

2 Methods

Finite Difference: I decided to use a centered-difference method because it gives a better estimate for the derivative than a forward difference method does. The step size was set to $h = 1 \times 10^{-8}$ because this performed well when checked against the finite differencing inside fmincon(). It didn't always match the derivatives calculated internally within fmincon(), but I suspect that that is due to fmincon dynamically changing the step size throughout the optimization.[2]

Complex Step: For complex step, the step size was chosen to be $h = 1 \times 10^{-30}$ because this was discussed in class as a typical value.

Algorithmic Differentiation: I used a MATLAB package called AutoDiff_R2016b.[3] This was initially difficult to work with but ultimately became easy to use.

Starting Point: The starting point for these optimizations has each of the cross-sectional areas equal to one square meter. All derivatives are calculated using this as the input and the optimization example is performed with this as the starting point.

Calculating Error: The relative errors were calculated relative to the complex step method. I calculated two error metrics: RMS and RMS %. The RMS error is calculated by finding the differences between the method under test and the complex step method and then taking the root mean square of those differences. The RMS % error was calculated as

$$\text{RMS } \% = \text{rms}\left(\frac{\text{Derivative using Current Method} - \text{Derivative using Complex Step}}{\text{Derivative using Complex Step}} \times 100\right) \tag{1}$$

and is used as a metric to normalize the differences.

[1] https://github.com/Classes-MCA/MEEN575-Optimization/tree/main/Homework%203
[2] https://www.mathworks.com/help/optim/ug/fmincon.html
[3] https://www.mathworks.com/matlabcentral/fileexchange/61849-autodiff_r2016b

Calculating Evaluation Time: The evaluation time was calculated by using MATLAB's "tic" and "toc" commands inside the function for calculating the Jacobian, with the "tic" inserted at the beginning and the "toc" inserted at the end.

3 Results

3.1 Relative Errors

The relative errors are shown in Table 1. Both methods compare favorably with the complex step method.

Table 1: Errors for the finite difference and algorithmic differentiation techniques relative to the complex step method. The RMS value is the RMS of the differences. The RMS % value is is the percent rms error relative to the actual derivatives.

		RMS	RMS %
Finite Difference	**Objective**	3.25 E -7	6.71 E -7
	Constraint	6.14	2.67 E -3
Algorithmic Differentiation	**Objective**	0	0
	Constraint	1.02 E -7	2.91 E -12

3.2 Evaluation Time

The different methods each take different amounts of time to run. Table 2 shows the differences in run times for each method. When this experiment is repeated, the run times do change, but the overall trends are generally preserved from run to run. The complex step method comes in as the fastest of these approaches, and is generally about twice as fast as standard finite differencing, which requires twice as many function calls as complex step. Both methods are 2-3 orders of magnitude faster than algorithmic differentiation.

Table 2: The time required to run each method. When run multiple times, the time for each evaluation varied, but the overall trends remain the same between runs.

		Evaluation Time (s)
Finite Difference	**Objective**	0.0926
	Constraint	0.0093
Complex Step	**Objective**	0.0366
	Constraint	0.0055
Algorithmic Differentiation	**Objective**	2.7398
	Constraint	2.1092

3.3 Using These Derivatives in Optimization

I chose to use the complex step method for the actual optimization. The convergence of the first-order optimality is shown in Figure 1. The first-order optimality is shown as a function of the number of function calls to the truss function, and the total number of function calls required to converge was 231. This is a satisfactory result, showing that the complex step method can be used to provide derivatives to the optimizer and provide a converged result.

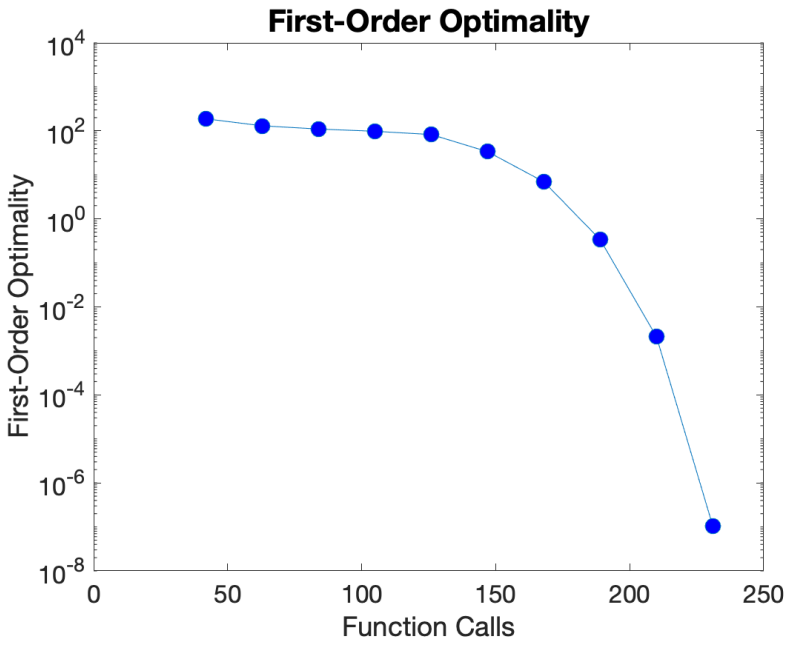

Figure 1: The convergence of the first-order optimality for the optimization using complex step.

4 Conclusion

4.1 Benefits of Each Method

Finite Difference: This can be done on any machine that can do all of the operations involved. One benefit in particular is that it can be run on machines that do not handle complex numbers.

Complex Step: This is the fastest approach studied here. It runs about twice as fast as finite differencing because it requires half as many function calls. It is also still very simple to evaluate and can return values with machine precision.

Algorithmic Differentiation: This can be run on machines that do not support complex numbers and still retain the ability to provide numerically-exact derivatives.

4.2 Drawbacks of Each Method

Finite Difference: This inherently involves subtracting two similar numbers, resulting in a loss of precision. As a result, it cannot return numerically-exact derivatives (results in Table 1 show errors at about half machine-precision). It is also slower than complex step.

Complex Step: This method can only be run on machines that know how to handle complex numbers. Some changes to the objective function had to be made to make it work properly.

Algorithmic Differentiation: This method was 2-3 orders of magnitude slower than the other two methods. This may be simply the implementation used in the MATLAB toolbox, and this method may in fact run just as fast as the other methods under different implementations.

4.3 Overall Takeaways

The complex step method comes out of this as the obvious winner of the three methods for computing derivatives. It provided faster and more accurate results than standard finite differencing, and in a fraction of the time required for algorithmic differentiation. It also worked well in the actual optimization, where it was able to provide a proper convergence.

References

[1] Martins, J. R. R. A., & Ning, A. (2021). Engineering Design Optimization.
http://flowlab.groups.et.byu.net/mdobook.pdf