Construction and Evaluation of an Incremental Iterative Version of a Parallel Multigrid CFD Code via Automatic Differentiation for Shape Optimization

Alan Carle and Mike Fagan

CRPC-TR98738
February 1998

Center for Research on Parallel Computation
Rice University
6100 South Main Street
CRPC - MS 41
Houston, TX 77005

Submitted February 1998

Construction and Evaluation of an Incremental Iterative Version of a Parallel Multigrid CFD Code via Automatic Differentiation for Shape Optimization*

CRPC-TR98738

Alan Carle and Mike Fagan

February 4, 1998

Abstract

Automatic differentiation (AD) is a technique for augmenting computer codes to compute derivatives of a subset of their outputs with respect to a subset of their inputs. AD has been shown to provide accurate, but inefficient, sensitivity-enhanced CFD codes for use in aerodynamic shape optimization. To address the inefficiency problem, several special purpose techniques have been suggested. One such technique is the incremental iterative (II) method. This report describes and evaluates the use of the II method in AD-augmented code for OVERFLOW, a parallel, multigrid CFD code.

1 Introduction

Shape optimization is an important part of modern aerodynamic design. An important component of the shape optimization process is the computation of derivatives of the aerodynamic outputs of computational fluid dynamics (CFD) programs with respect to shape parameters. In contrast with finite differences, automatic differentiation (AD), a technique for augmenting computer codes to compute derivatives of a subset of their outputs with respect to a subset of their inputs (see [4]), has been shown to deliver accurate derivatives of CFD codes [1, 7, 6]. Unfortunately, CFD codes augmented by AD using the simple "black box" (BB) approach, tend to take about the same or more time as two-sided finite differences to compute derivatives. Fortunately, it is often possible to combine AD with mathematical insight to generate efficient derivative codes. For CFD, one such mathematically-motivated technique is the *incremental iterative* (II) method. The details of the II method appear elsewhere in the literature [5, 7]. The salient points, however, are that II methods eliminate portions of the AD-generated derivative computations that are not required for well-converged CFD solutions, and that the methods can be implemented through a combination of AD and a small amount of human intervention.[1]

In light of these considerations, the task presented to us by NASA was to construct an II version of OVERFLOW 1.7r [3] and to compare its accuracy and efficiency to finite differences and to the BB version of OVERFLOW. OVERFLOW 1.7r is a parallel, multizone, multigrid flow solver being used for shape optimization at NASA. The use of parallelism and multigrid adds an element of complexity to the application of the II method to OVERFLOW.

In this report, we refer to the II version of OVERFLOW as OVERFLOW-ADII, or just ADII when it is clear that we are referring to OVERFLOW. OVERFLOW-ADBB, or just ADBB, is used to refer to the BB

*This work was supported by the National Aeronautics and Space Administration under Cooperative Agreement No. NCC 1 236, and by the National Science Foundation, through the Center for Research on Parallel Computation, under Cooperative Agreement No. CCR-9120008. Additional computing resources provided by NASA Ames High Performance Computing and Communication Program Testbed 1 Cooperative Research Agreement (through IBM P.O. # 835600Z-E1).

[1]We are, however, investigating methods to further reduce the amount of human intervention currently required for II construction.

version of OVERFLOW. For symmetry, OVERFLOW-FD, or just FD, is used to refer to the entire sequence of steps required to use finite differences to approximate derivatives of OVERFLOW.

The target hardware for this effort is the IBM SP2. Most experiments were run on Rice's 8 node SP2, but some were run on the SP2 at NASA Langley, and a few confirming runs were conducted on Rice's 32-processor Hewlett-Packard/Convex SPP-2000 X-Class Exemplar.

For this project, we used two test cases.[2] The first test case (2Z) consists of two zones with dimensions 271x91x3 and 61x225x3. The second test case (3Z) consists of three zones with grid dimensions 181x61x3, 481x121x3 and 261x61x3. These grids are both "pseudo 3-D" grids, i.e., each zone consists of three identical 2-D planes. For both of our test cases, the provided control files specify 3 levels of multigrid, and the use of the "full multigrid" feature of OVERFLOW, which we describe next.

Full multigrid, for 3 levels of multigrid, first executes a series of degenerate V-cycle solution steps on the coarse grid and interpolates the resulting coarse resolution flow field to the medium grid. It then executes a series of V-cycle steps on the medium and coarse grids and interpolates the resulting medium resolution flow field to the fine grid. It then executes all of the remaining V-cycles on the fine, medium and coarse grids. In this report, we refer to the coarse and medium grid preliminary V-cycle steps as the multigrid "prefix." In the rest of this report, we use the OVERFLOW 1.7r variable names FMG and NSTEPS to describe multigrid solution protocols. For example, a test case using a 3-level full multigrid protocol with a prefix consisting of 50 coarse grid steps and 50 medium grid steps, followed by 200 fine grid steps for a total of 300 steps, would be indicated as "FMG = 50,50, NSTEPS=200." We use FMG=0 to indicate a multigrid solution protocol that does *not* use the full multigrid feature, i.e., all steps are V-cycles on the finest grid.

This report is organized as follows: Section 2 reviews the mathematics underlying the II method and describes our approach to producing OVERFLOW-ADII from OVERFLOW 1.7r. Section 3 gives the results of our derivative validation study. Section 4 compares the performance of FD, ADBB, and ADII. Section 5 summarizes our conclusions.

2 The II Method

The II method can be used to compute derivatives for programs that solve equations by iterating a recurrence until some associated residual quantity is "small enough." Many scientific programs fall into this category. In the following discussion, we give an overview of the II method. A more complete (and rigorous) discussion can be found in [5] or [7].

A prototypical recurrence iteration, expressed mathematically is

$$L_n \Delta f_n = R_n$$

where f_n is a current solution estimate, R_n is a residual difference between current and desired aspects of the solution, and L_n is the current linearization of the problem operator. The n-th iteration computes both the residual R_n and the linear operator L_n, and then solves the linear system for Δf_n. This Δf_n is added to the current solution f_n to yield the new solution candidate. This process normally terminates when Δf_n is "small enough."

Differentiating the iterative process yields another iterative process whose iteration is

$$L_n \Delta f'_n + L'_n \Delta f_n = R'_n.$$

Now suppose that we have iterated a number of times, and f has essentially converged. Under this condition, Δf will be "small." Furthermore, L_n will change "very little" once f has converged. If Δf is "very small" relative to L'_n, then the second term in the sum is close to 0, and may be dropped from the calculation. Likewise, L_n is nearly constant, so the derivative recurrence can be simplified to

$$L_* \Delta f'_n = R'_n$$

where L_* indicates the value of L when f has converged. Note the similarity of this simplified recurrence to the original undifferentiated recurrence.

[2]We extend special thanks to Pieter Buning for building the grids and providing the control files, and to Ray Gomez for suggesting the 3 zone case.

This simplified method for computing derivatives has several advantages. First, since there is no need to compute $L'_n \Delta f_n$, the simplified recurrence calculation should be more efficient than the original derivative recurrence. Second, the code in the original program that is used to solve $L\Delta f$ can be used to solve $L_* \Delta f'_n$. Since the original solver may have been hand tuned, reusing the original solver for derivative calculations *may* have a significant impact on the performance of the derivative code.

Original Iteration

```
read inputs $I$
initialize $f_0$
do $n = 0 \ldots$
    $R_n$ = residual($I, f_n$)
    $\Delta f_n$ = solver($I, f_n, R_n$)
    $f_{n+1} = f_n + \Delta f_n$
enddo
compute outputs $D(I, f_*)$
```

BB Iteration

```
read inputs $I, I'$
initialize $f_0, f'_0$
do $n = 0 \ldots$
    $R_n, R'_n$ =
        g_residual($I, I', f_n, f'_n$)
    $\Delta f_n, \Delta f'_n$ =
        g_solve($I, I', f_n, f'_n, R_n, R'_n$)
    $f_{n+1} = f_n + \Delta f_n$
    $f'_{n+1} = f'_n + \Delta f'_n$
enddo
compute outputs
    $D(I, f_*)$,
    $D'(I, I', f_*, f'_*)$
```

II Iteration

```
Given $f_*, f'_0$
read inputs $I, I'$
do $n = 0 \ldots$
    $R_n, R'_n$ =
        g_residual($I, I', f_*, f'_n$)
    for each $v$ in $I$
        $\Delta f'_n(v)$ =
            solver($I', f_*, R'_n(v)$)
    endfor
    $f'_{n+1} = f'_n + \Delta f'_n$
enddo
compute outputs
    $D(I, f_*)$,
    $D'(I, I', f_*, f'_*)$
```

Figure 1: Code Templates for II Processing.

In Figure 2, we present code templates for use in applying the II method to a CFD code. In the original iteration, the independent variables I are used to construct the initial flow field f_0, which is then iteratively updated to the final flow field f_*. The dependent variables D are then computed using the final flow field f_*. Applying AD to the original iteration generates the BB version of the iteration which computes both D and D'. To create the II iteration from the BB version, assuming that we are provided with the converged solution f_* computed by the original iteration, f'_0, an initial starting point for the iteration computing f'_*, we delete the f_n update and then we replace the call to g_solve with an explicit loop over each of the independent variables. The challenge for multigrid is to identify the f_n updates and interpolation steps for each level of the grid. The loop invokes the original solver on each component of R'_n to compute the derivative update component.

We used Adifor 2.0 [2] to differentiate OVERFLOW to create ADBB, and then modified the source code for the ADBB iteration to create the code for the II iteration. We then constructed ADII using the original OVERFLOW code to compute f_*, ADBB to create the initial values for f'_0, and the code for the II iteration to compute f'_* from f_* and f'_0. ADII consists of the following three step sequence of operations:

Step 1. Use ADBB to execute the full multigrid prefix, and 1 additional fine grid V-cycle to generate the flow field f_{prefix} and its derivative f'_{prefix}. f_{prefix} will be used as f_0 in Step 2. f'_{prefix} will be used as f'_0 in Step 3.

Step 2. Using OVERFLOW's restart facility, run OVERFLOW on the remaining fine grid V-cycles starting with f_0, and converge to the final flow field f_*.

Step 3. Run the II iteration with f_*, f'_0 to compute f'_*.

3 Validation

We began our evaluation of ADII by looking at the derivatives computed by ADII, ADBB and FD. We considered two sets of derivatives: derivatives with respect to angle of attack (`alpha` variable in OVERFLOW), and derivatives with respect to geometric variables (`x,y,z` grid variables in OVERFLOW). For the geometric derivative study, we simply perturbed the location of a single "X" coordinate in the input grid. Of the 13 aerodynamic outputs computed by OVERFLOW, our test case providers suggested we monitor `cmp`, the pitching moment coefficient most closely.[3] Hence, all the plots shown are derivatives of `cmp` (with respect to either `alpha` or the chosen coordinate). We performed all of these validations on the 2Z test case. For the 3Z test case, we limited our validation to angle of attack derivatives, and only compared the ADII and ADBB results.

3.1 Comparison of ADBB and FD

The comparison of ADBB and FD, shows good agreement for both angle of attack and geometric derivatives. We give a sample of our results here. The results not shown are similar.

Comparison of ADBB and FD, angle of attack, 2Z. Figure 2 shows the agreement of ADBB with FD for the derivative of `cmp` with respect to `alpha`, using 1.0E-04 as the finite difference perturbation. In this case, FMG=50,50, NSTEPS=600 for ADBB and for each OVERFLOW run for FD. The finite difference approximations were computed using one-sided forward differences.

Comparison of ADBB and FD, geometric, 2Z. Figure 3 shows the agreement of ADBB and FD for the derivative of `cmp` with respect to the location of a single "X" coordinate, using 1.0E-06 as the finite difference perturbation. For this case, FMG=50,50, NSTEPS=600. Finite difference approximations are one-sided forward differences. We note in passing that geometric derivatives done this way are quite sensitive to perturbation size. A number of perturbations did *not* work. Caveat finite differencor.

3.2 Comparison of ADBB and ADII

The accuracy of the IIM derivative calculation is not so clear, partly because there are so many different degrees of freedom in the IIM protocol. The three most obvious (to us) protocol parameters are:

1. The number of steps taken to converge the starting flow field.
2. The number of steps of ADBB to use as a "baseline."
3. The number of ADII steps taken to converge the IIM derivatives.

In our studies, we chose to take the number of steps to converge the function value as our choice of ADBB baseline, eliminating one degree of freedom. The remainder of this section details our experiments.

Comparison of ADBB and ADII, angle of attack, 2Z, experiment 1. Our initial choice for baseline function convergence was provided to us with the grids. For the 2Z case, we had FMG=50,50, NSTEPS=200. Figure 5 shows the derivative of `cmp` with respect to `alpha`. The figure shows that the convergence of ADII, even after 300 steps is not especially good. Furthermore, taking *more* than 300 steps has badly divergent behavior, as shown in Figure 6. This divergence suggested that perhaps the flow field might not have been as converged as we thought. Figure 4 shows the behavior of `cmp` for the first 300 steps. In spite of the apparent convergence, the poor behavior of ADII mandated a second experiment, using a flow field that had been run for more steps.

Comparison of ADBB and ADII, angle of attack, 2Z, experiment 2. For our second ADII study, we ran the function and ADBB as FMG=50,50, NSTEPS=600. In this case, the convergence appears to be somewhat better than the convergence exhibited in experiment 1. Furthermore, derivatives of fair accuracy can be obtained in far less than 700 ADII steps. Table 1 shows the relative accuracy of various derivative outputs as a function of the number of ADII steps. The relative accuracies in the table are expressed as the percentage of the ADBB value at step 700. Unshown aerodynamic derivatives have small magnitudes (less than 1.0×10^{-18}). The numerical column headings indicate the number of ADII steps taken. The table indicates that the convergence is not uniformly strong Figure 7 compares the ADII and ADBB behavior of the `cmp` output for `alpha` derivatives.

[3]Again, thanks to Ray Gomez, Pieter Buning, and Larry Green for their sharing their expertise.

	ADII Steps					
	200	300	400	500	600	700
clp	0.079	0.024	0.004	0.014	0.018	0.075
cdp	1.06	0.984	0.932	0.892	0.821	0.358
clf	7.63	8.457	8.935	9.41	9.94	10.4
cdf	0.577	3.082	7.03	12.6	20.4	30.4
cmp	0.0299	0.128	0.0632	0.0462	0.118	0.278

Table 1: Relative accuracy (percent) of ADII derivatives with respect to `alpha`

Comparison of ADBB and ADII, geometric, 2Z. Since ADII showed better behavior for FMG=50,50, NSTEPS=600 on the `alpha` case, our next verification exercise examined the geometric derivatives. Again, based on our examination of the table, we rate the convergence as "fair." As in the previous table, Table 2 shows the relative accuracies for this exercise.

	ADII Steps					
	200	300	400	500	600	700
clp	2.07	2.18	2.09	1.95	1.77	2.01
cdp	7.67	8.35	9.16	10.3	11.8	8.06
clf	5.96	5.52	5.96	7.00	9.15	13.1
cdf	15.9	15.1	13.5	12.1	10.1	0.355
cmp	2.04	2.15	2.08	1.95	1.80	1.77

Table 2: Relative accuracy (percent) of ADII geometric derivatives

Figure 8 compares the ADII and ADBB behavior of the `cmp` output for geometric derivatives.

Comparison of ADBB and ADII, angle of attack, 3Z. A comparison of ADII and ADBB for the 3Z test case behaves much like the 2Z case. For this test case, we used FMG=50,50, NSTEPS=400 for the derivative protocol Figure 10 compares `cmp` derivatives with respect to `alpha`, as computed by both ADII and ADBB. Again, we rate the convergence as "fair."

One more anomaly. While the behavior of the FMG=50,50, NSTEPS=600 run is "fair," we were still concerned that we did not have a sufficiently converged flow field to yield good answers. Consequently, we made one ADBB run of FMG=50,50, NSTEPS=3000. Figure 9 shows the behavior of the `cmp` derivative with respect to `alpha`. As the figure shows, there are two distinct plateaus — either one of which might be a presumed stationary point. We have no explanation for this anomaly. Our best guess is that the problem itself has some unsteadiness.[4]

4 Performance

In light of our insecurity about the accuracy of the ADII method for the provided test case, we conducted only limited timing studies. The results in this section are based on the FMG=50,50, NSTEPS=200 angle of attack results. For illustrative purposes, we have extrapolated some timing numbers for different protocols.

Timings for the computation of the derivative of `cmp` with respect to `alpha` for 2Z test case ADBB, ADII and (one-sided) FD are shown in Table 3. These numbers indicate that ADII does perform better than

[4] We also considered the possibility that we had not correctly implemented the ADII method. We validated our implementation in three separate ways. First, we implemented the ADII using two distinct algorithms and verified that both gave the same answers. Second, we verified that the flow field and its partial derivatives remained constant for each ADII iteration. Finally, we iterated the derivative term excluded from the ADII, and discovered that the excluded term was behaving like the ADII, but with opposite sign. All of these results suggest a correct ADII implementation.

Program	Total time (seconds)
ADBB	7960
ADII	5102
FD	2293.47

Table 3: Timings for computation of the derivative of `cmp` with respect to `alpha` for 2Z test case.

ADBB, but it still performs rather poorly compared to FD. We also generated ADBB and ADII code using a feature of Adifor 2.0 that generates derivative codes that compute a single derivative at a time. Adifor normally generates derivative code that accommodates more than one derivative per run. By default, these derivatives are stored in an array and computed by looping over each of the entries in the array. To compute a single derivative, it is sufficient to replace the arrays with scalars and then eliminate the loops. We refer to the scalar versions of ADBB and ADII, as ADSBB and ADSII, respectively. As shown in Table 4, the scalar versions significantly outperform their vector counterparts. Unfortunately, ADSII performs more poorly than ADSBB.

Program	Total time (seconds)
ADSBB	2605
ADSII	3457

Table 4: Additional timings for computation of derivative of `cmp` with respect to `alpha` for 2Z test case.

The timings indicated, however, reflect total time for an ADII computation. That is, the time for the prefix, the function convergence, and the IIM steps themselves are all included in the total timing for ADSII. Just looking at the cost of an ADSII step versus ADSBB gives:

Program	Average step time (seconds)
ADSBB	11.748
ADSII	10.296

So, an ADSII portion is still marginally faster than an ADSBB.

If we amortize the cost of computing the converged flow field, then SII iteration can be better than SBB but is still slower than FD. When the process requires more than one derivative, the computation of the converged flow field needs to be done *once* per set of derivatives, since the same converged flow field is required for each SII derivative computation. Hence, the cost of the function evaluation can be amortized over the cost of multiple derivatives.

Real improvements over FD, however, will require taking fewer steps. To illustrate the potential benefit of stopping early, we extrapolate the timing for experiment 2 of Section 3. Using our timing results, we estimate that total ADSBB time for FMG=50,50, NSTEPS=600 would be 267 (prefix) $+ 599 \times 11.748$ (ADBB steps) = 7304 seconds. Using the FD value from Table 3, we estimate FD time for this case to be 6869 seconds. Table 5 shows the estimated time (in seconds) and maximum relative error (in percent) as a function of the number of II iterations. In light of these estimates, if ADII can be stopped after 200 or 300 steps, then it will take less total time than FD.[5]

	ADII Steps				
	200	300	400	500	600
time(seconds)	5750.4	6780.0	7809.6	8839.2	9868.8
max rel error(%)	7.63	8.457	8.935	12.6	20.4

Table 5: Estimated performance and accuracy for ADSII

[5] We are aware that our FD algorithm is naive. FD in practice would probably use restarts to further reduce the computation time. We intend to examine more sophisticated FD techniques in future work.

5 Conclusions

Our main concern in this study was accuracy. As detailed in Section 3, the ADII method does not exhibit strong convergence for the test cases we had. A (presumably) more converged flow field did seem to have a positive effect on ADII convergence, but the favorable effect was not overwhelming. Furthermore, running too many steps of ADII results in divergence. On the positive side, the ADII does appear to reach its "quasi-converged" plateau in fewer steps than ADBB.

Our timing results indicate that ADII by itself does not lead to any significant improvement in the speed of derivative calculation. The ADII steps themselves are marginally faster, but are still not competitive with finite differences. If, however, the accelerated convergence can be consistently realized, then the ADII method can be competitive with finite differences.

References

[1] C. Bischof, T. Knauff, L. Green, and K. Haigler. Parallel calculation of sensitivity derivatives for aircraft design using automatic differentiation. In *5th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, pages 73–86, 1994. AIAA-94-4261-CP.

[2] Christian Bischof, Alan Carle, Peyvand Khademi, and Andrew Mauer. Adifor 2.0: Automatic differentiation of Fortran 77 programs. *IEEE Computational Science and Engineering*, 3(3):18–32, Fall 1996.

[3] Pieter C. Buning, Dennis Jespersen, Thomas Pulliam, William M. Chan, Jeffery P. Slotnick, Steven E. Krist, and Kevin J. Renze. *OVERFLOW User's Manual, version 1.7r*, October 1996.

[4] A. Griewank. On automatic differentiation. In M. Iri and K. Tanabe, editors, *Mathematical Programming: Recent Developments and Applications*, pages 83–108. Kluwer Academic Publishers, 1989.

[5] Andreas Griewank, Christian Bischof, George Corliss, Alan Carle, and Karen Williamson. Derivative convergence for iterative equation solvers. *Optimization Methods and Software*, 2:321–355, 1993.

[6] J. Issac and R. Kapania. Aeroelastic sensitivity analysis of wings using automatic differentiation. In *6th AIAA/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, pages 1176–1186, 1996. AIAA-96-4119-CP.

[7] V. Korivi, L. Sherman, A. Taylor, G. Hou, L. Green, and P. Newman. First- and second-order aerodynamic sensitivity derivatves via automatic differentiation with incremental iterative methods. In *5th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, pages 87–120, 1994. AIAA-94-4262-CP.

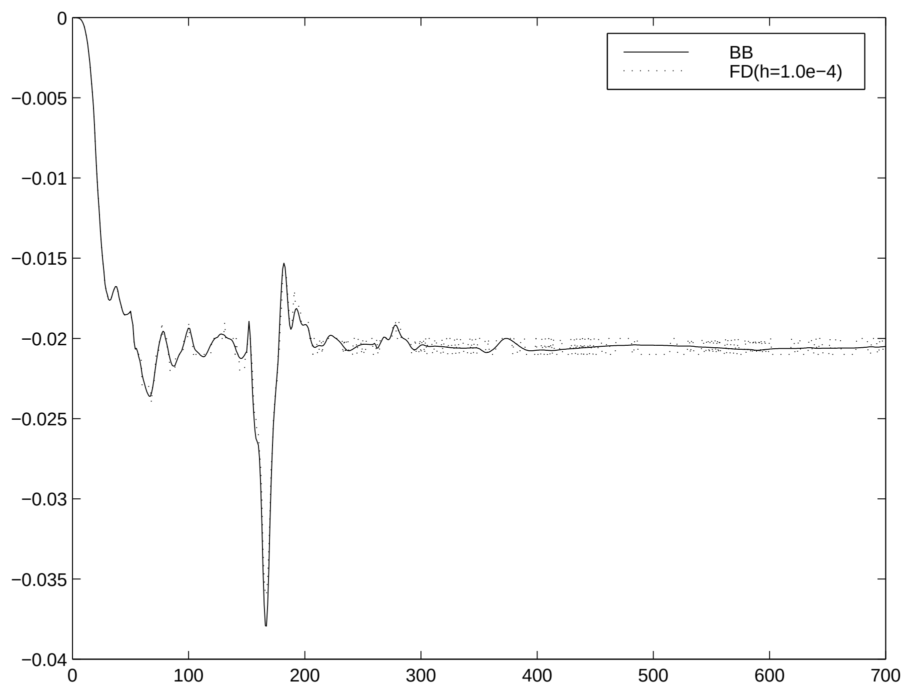

Figure 2: Comparison of ADBB and FD derivatives of `cmp` with respect to `alpha`, 2Z.

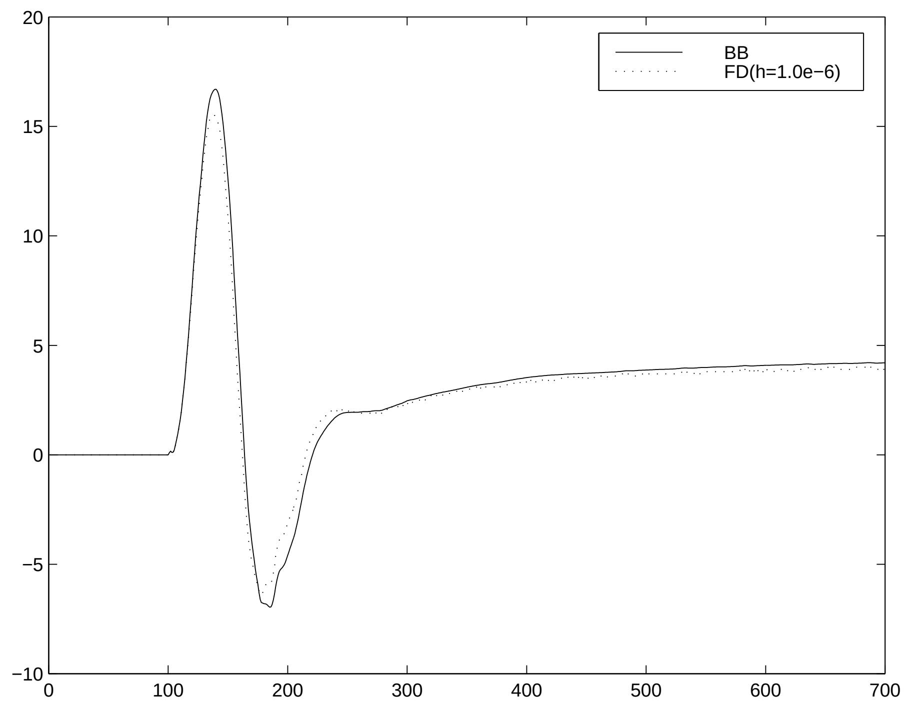

Figure 3: Comparison of ADBB and FD geometric derivatives of `cmp`, 2Z.

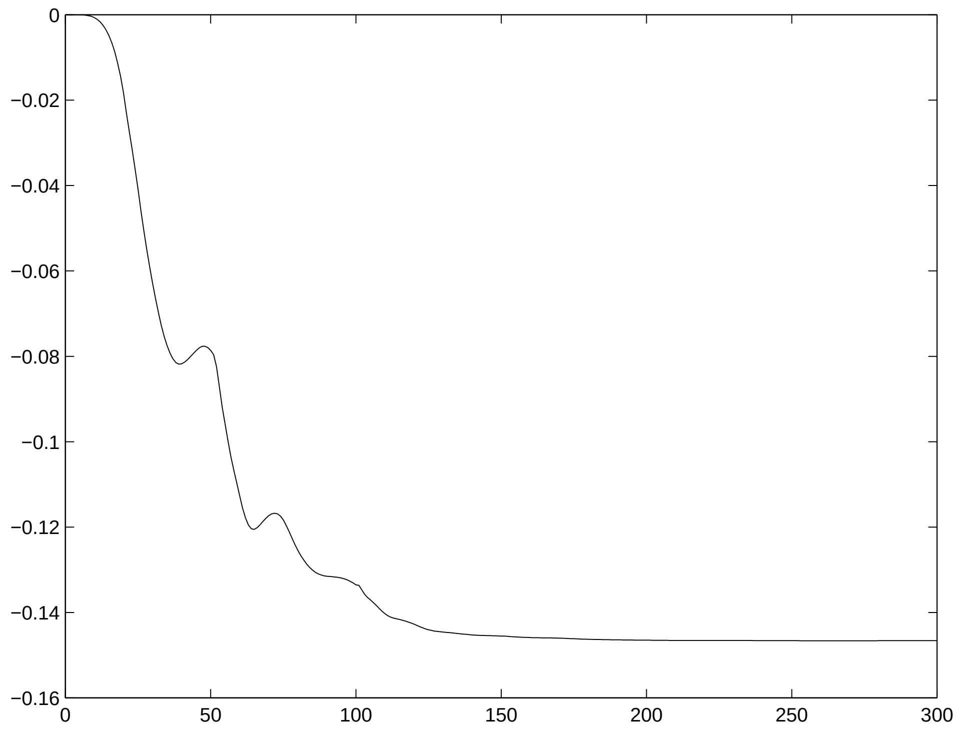

Figure 4: Aerodynamic output `cmp`, 2Z.

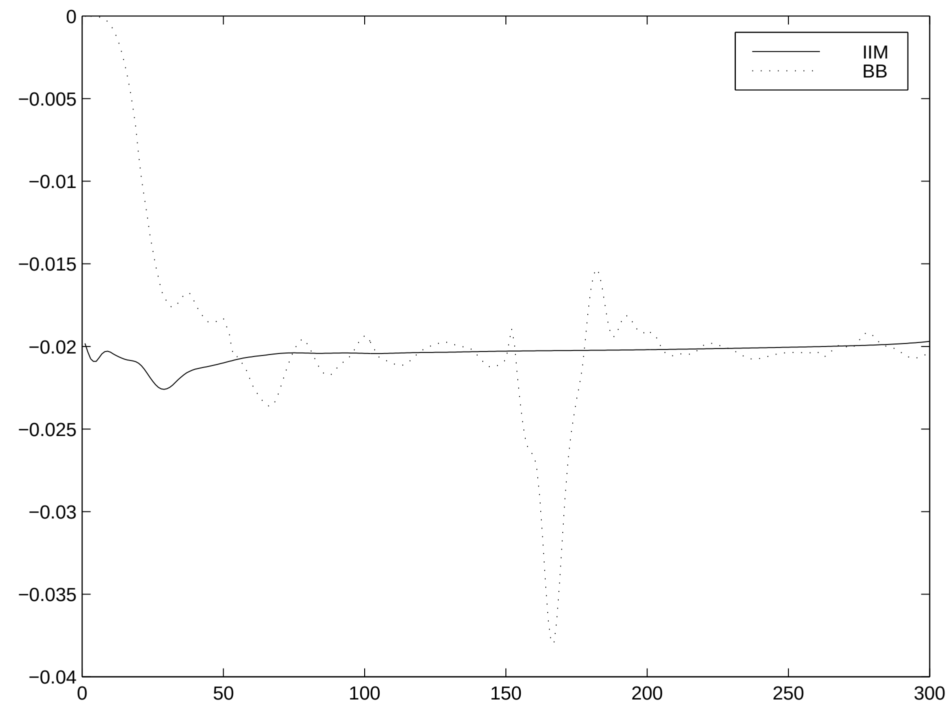

Figure 5: Comparison of ADBB and ADII derivatives of `cmp` with respect to `alpha`, FMG=50,50, NSTEPS=200, 2Z.

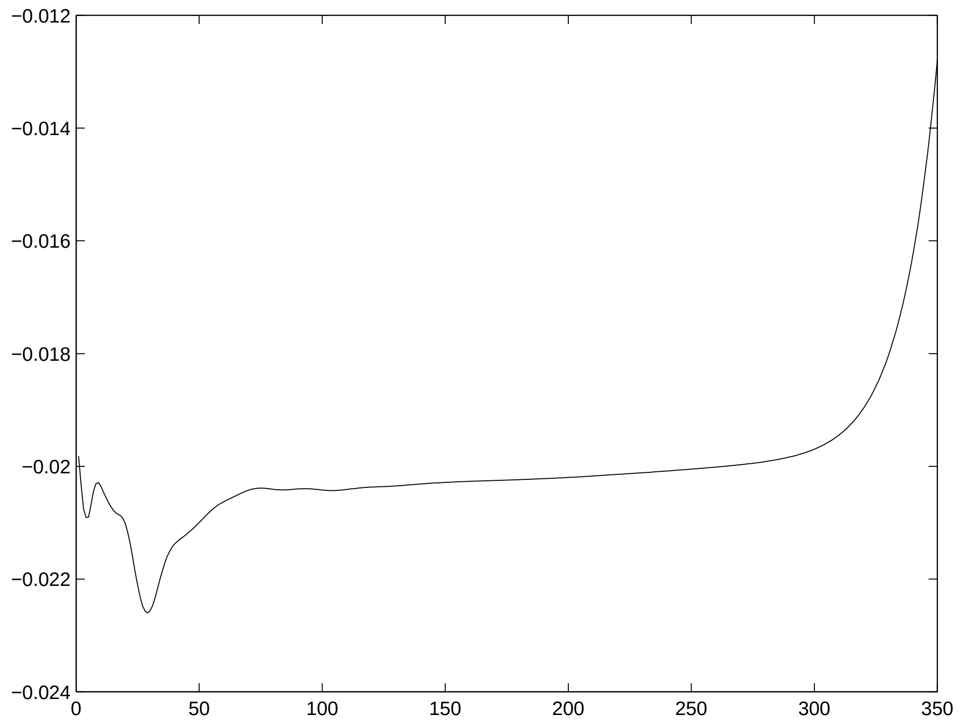

Figure 6: Divergence of ADII derivative of `cmp` with respect to `alpha`, 2Z.

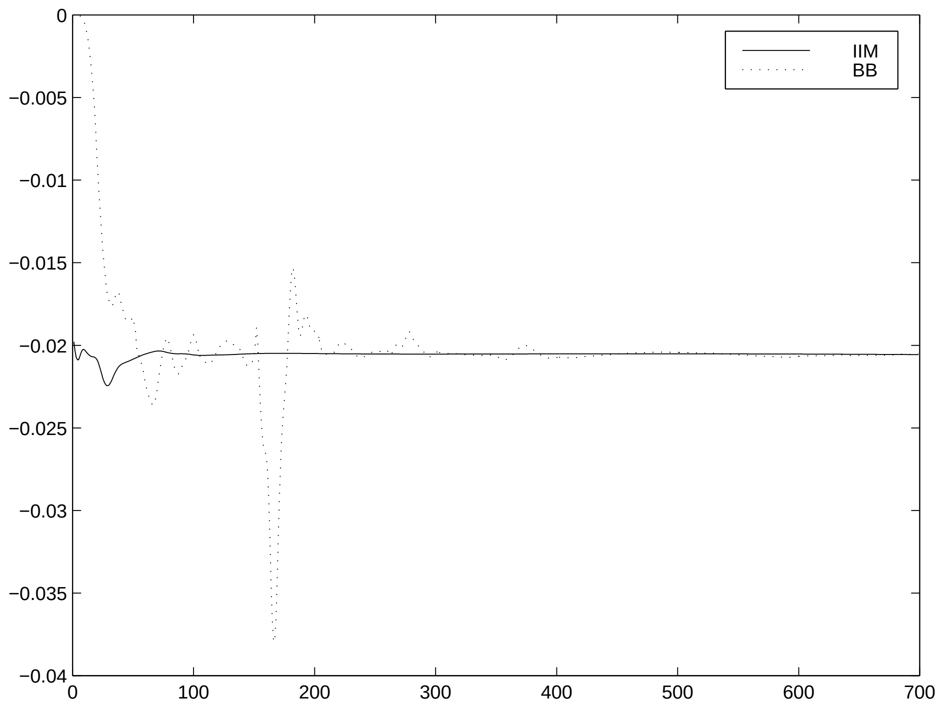

Figure 7: Comparison of ADBB and ADII derivatives of `cmp` with respect to `alpha`, FMG=50,50, NSTEPS=600, 2Z.

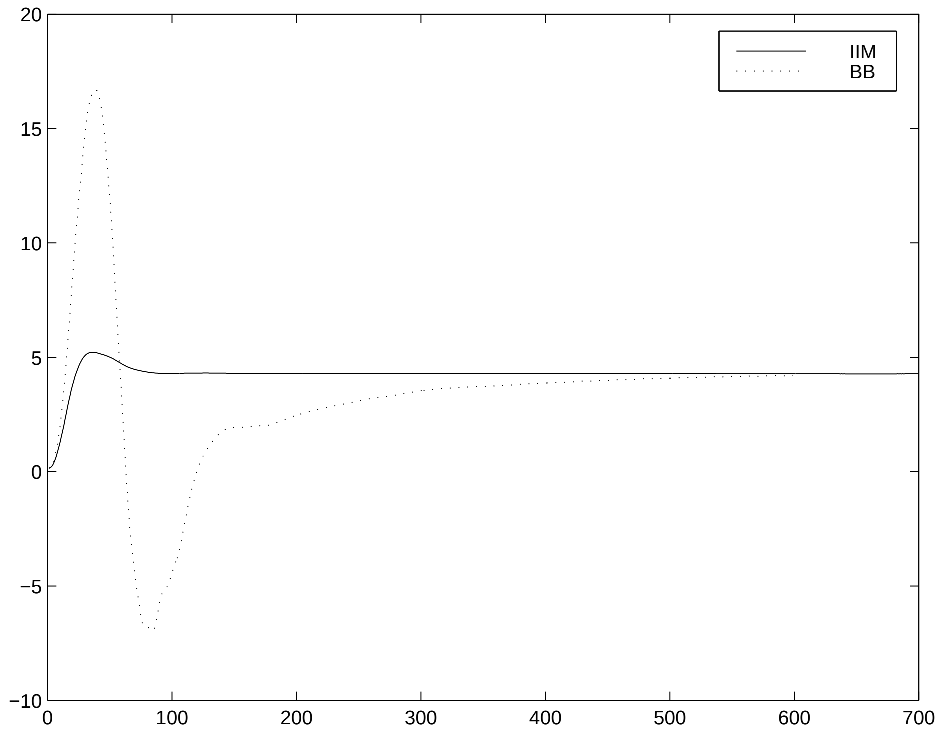

Figure 8: Comparison of ADBB and ADII geometric derivatives of `cmp`, FMG=50,50, NSTEPS=600, 2Z.

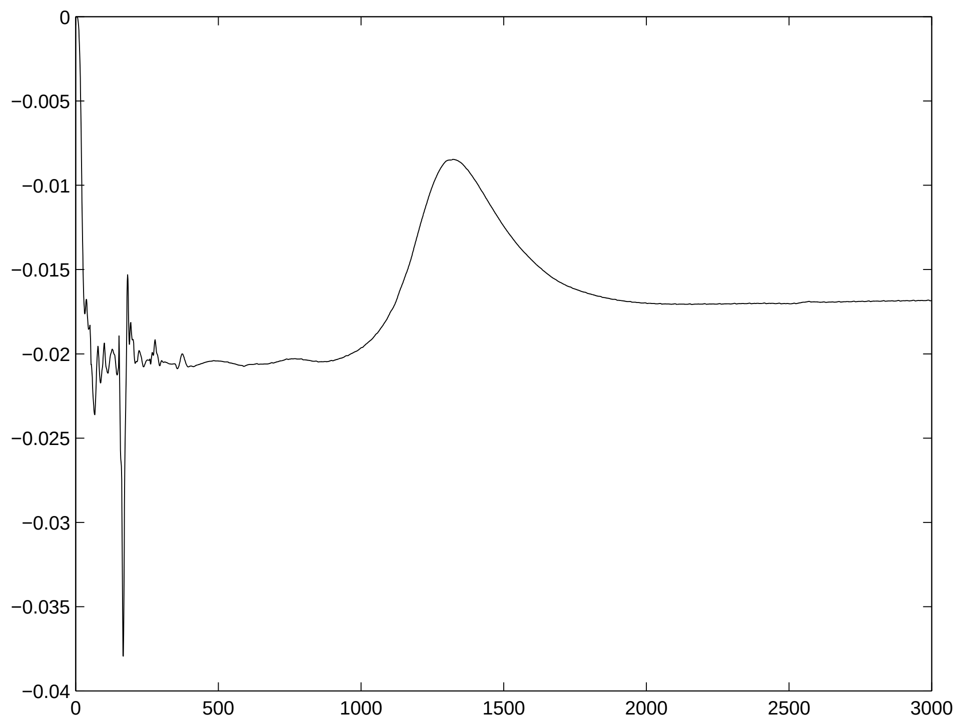

Figure 9: Derivative of `cmp` with respect to `alpha`, FMG=50,50, NSTEPS=3000, 2Z.

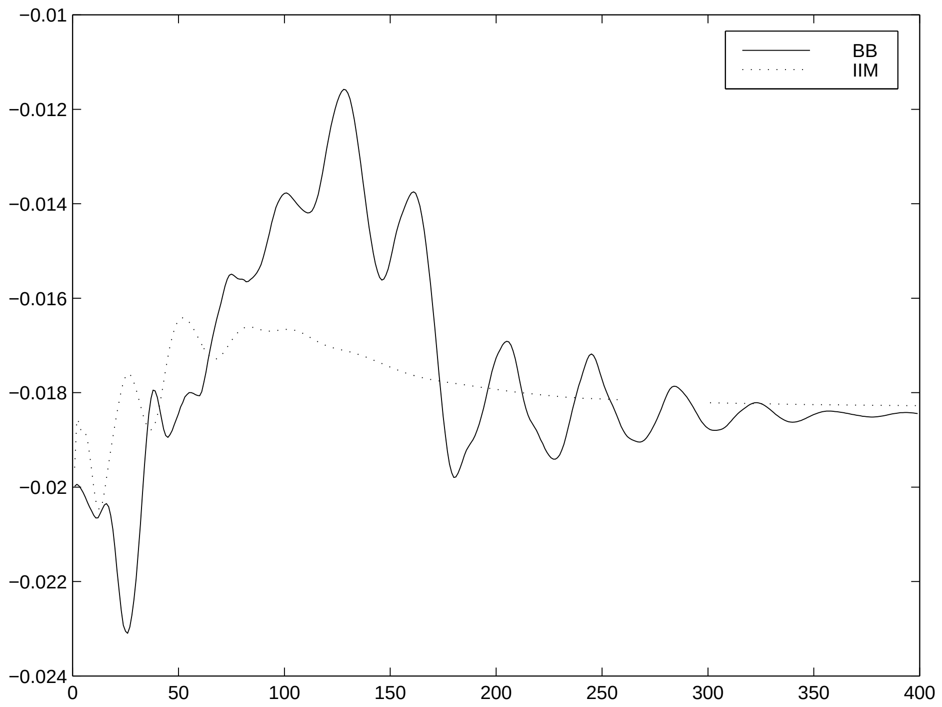

Figure 10: Comparison of ADBB and ADII derivatives of `cmp` with respect to `alpha`, 3Z.