

Adjoint Differentiation of a Structural Dynamics Solver

Mohamed Tadjouddine^a, Shaun A. Forth^a and Andy J. Keane^b

^a Engineering Systems Department, Cranfield University (Shrivenham Campus), Swindon SN6 8LA, UK.

^b Computational Engineering Design Centre, University of Southampton, Southampton SO17 1BJ, UK.

Abstract

The design of a satellite boom using passive vibration control by Keane [J. of Sound and Vibration, 1995, 185(3),441-453] has previously been carried out using an energy function of the design geometry aimed at minimising mechanical noise and vibrations. To minimise this cost function, a Genetic Algorithm (GA) was used, enabling modification of the initial geometry for a better design. To improve efficiency, it is proposed to couple the GA with a local search method involving the gradient of the cost function. In this paper, we detail the generation of an adjoint solver by automatic differentiation via ADIFOR 3.0. This has resulted in a gradient code that runs in 8.2 times the time of the function evaluation. This should reduce the rather time-consuming process (over 20 days by using parallel processing) of the GA optimiser for this problem.

1 Introduction

This paper details the differentiation of a computer code BEAM3D [9, 10], which models the structural dynamics of a three-dimensional satellite boom based on the receptance theory whereby the behaviour of the global structure is predicted from the Green functions of the individual components, evaluated as summations over their mode shape. Typically, the structure, as pictured in Figure 1, is excited by a point transverse force near an end beam and the energy level is measured at the opposite end beam. To minimise vibrations and noise, the geometry of the structure is modified so as the frequency average response of the satellite boom is improved. This is known as *passive vibration control* [5].

The satellite boom (see Figure 1) consisted of 90 Euler-Bernoulli beams all having the same properties per unit length. The three joints at the left hand end of the structure were fixed. The 90 beams are connected by 30 joints excluding the three fixed joints. For further details in the design of the structure, see [4, 6]. In previous work, the optimisation aimed at minimising the frequency average response of the end beam from which the energy is measured (in the range 150-250 Hz) by finding a superior design or geometry. For that purpose, Genetic Algorithms (GA) were used. GAs are known to work for fairly large cost with a strong chance to find the global minimum. Generally, GAs do not require the gradient of the cost function. However, the application of GAs in large scale industrial applications is limited due to the large number of expensive evaluations of the cost function. For our application, the first 10 generations for a population size of 100 took over 10 days to complete using parallel processing [4].

Attention has now shifted to hybrid genetic algorithm-local search combining Darwin and Lamarck's evolution models. Darwin's evolution is based on "natural selection" considering that the most fit individuals are likely to survive. Lamarck's evolution takes the view that individuals may improve within their environment as they became adapted to it and that the resulting changes are inheritable. Consequently, one forces the genotype of an improved individual to reflect the result of such

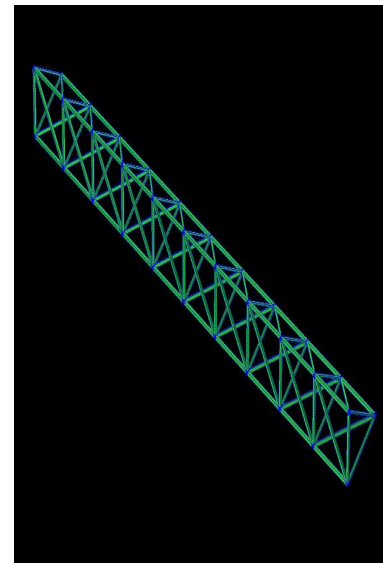


Figure 1: Initial geometry of a satellite boom

an improvement by replacing the individual into the population for competitive and reproductive opportunities [7]. In our application, the local search is done using a gradient method, which requires the calculation of the gradient of a cost function $\mathbf{F} : \mathbb{R}^{453} \rightarrow \mathbb{R}$ (453 independents representing the coordinates of the points in the design geometry). The computation of $\mathbf{F}(\mathbf{x})$ involves complex variable calculations, which are handled by ADIFOR 3.0 [1] since the dependents and independents are real values [8].

The gradient $\nabla\mathbf{F}$ of such a function is cheaply calculated using the reverse mode since the cost of the gradient is independent of the number of the independents. In fact, the cost of evaluating a gradient by reverse mode AD is bounded above by a small factor of the cost of evaluating the function [2]. A bottleneck may be the memory needed to run the reverse AD generated code. For the BEAM3D code, the AD reverse mode produces an adjoint code, which after being tuned manually for performance enhancement, calculated the function and its gradient in 8.2 times the CPU time required for its function evaluation. Moreover, that adjoint code runs 56.6 times faster than the one-sided finite-differences (FD) on a Sun Blade 1000 machine with 1200 MHz, 8 MB external cache, and 2 GB RAM.

2 Differentiation of the BEAM3D Code

The BEAM3D code starts by reading in data from files representing the boom geometry (listing of all beams and their connections) and certain properties of each beam. Given extra information such as the range of frequencies over which to solve, the number of data points, etc., the program builds up a linear complex system $\mathbf{A}\mathbf{u} = \mathbf{b}$ and solves it for each frequency. Prior to differentiation, the code is cleaned up so as data reading is done outside the subroutines to be differentiated. Furthermore, to make the code processed by ADIFOR 3.0, the code was rewritten according to Fortran 77 standard. The resulting computer code is differentiated using FD and AD via ADIFOR 3.0.

2.1 Initial Differentiation

We first computed a single directional derivative $\dot{y} = \nabla\mathbf{F}(\mathbf{x})\dot{\mathbf{x}}$ using FD and the AD forward mode and a single directional adjoint $\bar{x} = \nabla\mathbf{F}(\mathbf{x})^T\bar{\mathbf{y}}$ using the reverse mode AD. By definition of the adjoint operator, we have the following equality: $\langle \bar{\mathbf{x}}, \dot{\mathbf{x}} \rangle = \langle \bar{\mathbf{y}}, \dot{\mathbf{y}} \rangle$. This allows us to validate the results of the differentiation. The initial ADIFOR 3.0 generated codes gave inconsistent results with those from FD. This is caused partly by non-differentiable statements in the code for the function \mathbf{F} .

2.2 Dealing With Non-differentiability

A major assumption in the Automatic Differentiation approach is that the function \mathbf{F} to be differentiated is composed of elemental functions ϕ that are continuously differentiable on their open domains [2]. At a point on the boundary of an open domain, \mathbf{F} is continuous but $\nabla\mathbf{F}$ may jump to a finite value or even infinity. This becomes an issue when the computer code that represents the function contains branches, some kink functions (e.g. `abs`) or inverse functions (e.g. `sqrt`, `arctan`, or `arccos`). It is known that these pathological cases can be tackled by calculating derivatives in a given direction [2].

The first type of non-differentiability encountered in BEAM3D was due to a branching construct illustrated by the code fragment of Figure 2. As `xdiff`, `ydiff` are active variables, the differentiation of this code fragment gave point-valued derivatives that prevented the function \mathbf{F} from being differentiable. Such branches represent constraints on the design geometry and, in our case, may be safely removed.

The second type of non-differentiability is due to the presence of the functions `arccos` and

```

if (xdiff .eq. 0.0 .and. ydiff .eq. 0.0) then
  yor(1,i) = zdiff/beam_leng(i)
  yor(2,i) = 0.0
  yor(3,i) = 0.0
else ....

```

Figure 2: A code fragment testing whether an active variable is zero

`abs`. ADIFOR 3.0 allows us to locate possible non-differentiable points by running the derivative code with the *Exception Handling* turned on. This raised warnings concerning the functions `arccos` and `abs`. We located the part of the code, which caused such anomalies shown on the left of Figure 3. We then used algebra and trigonometric formula to rewrite some of the algebraic expressions containing (`arccos`, `abs`, `sin` and `tan`) as on the right of Figure 3. This transformation resulted in equivalent expressions calculating the same values but differentiable around the vicinity of their arguments.

```

l2=datan2(xdiff,zdiff)
m2=dsqrt(xdiff*xdiff+zdiff*zdiff)/beam_leng(i)
sgn1 = -1.0d0
sgn2 = -1.0d0
sgn3 = -1.0d0
if (xdiff.lt.0.0d0) sgn1 = -sgn1
if (ydiff.gt.0.0d0) sgn2 = -sgn2
if (zdiff.lt.0.0d0) sgn3 = -sgn3
yor(1,i)=sgn1*dabs(dsin(dacos(m2))*dsin(l2))
yor(2,i)=sgn2*dabs(m2)
yor(3,i)=sgn3*dabs(dsin(dacos(m2))*dcos(l2))

myt1=ydiff/beam_leng(i)
myt2=dsqrt(xdiff*xdiff+zdiff*zdiff)
=> yor(1,i)=myt1*(-xdiff/myt2)
yor(2,i)=myt2/beam_leng(i)
yor(3,i)=myt1*(-zdiff/myt2)

```

Figure 3: A code fragment (left) and its equivalent but differentiable (right)

Finally, the complex linear solver $\mathbf{A}\mathbf{u} = \mathbf{b}$ employs the LAPACK routine `zgesv`. Its differentiated routines by ADIFOR 3.0 gave inconsistent results with FD. We therefore hand-coded its derivative.

2.3 Complex Linear Solver

Instead of mechanically using ADIFOR 3.0 to differentiate the complex linear solver that dominates the CPU time, we hand-coded the forward and reverse codes. In the forward code, the following procedure is used:

1. Perform an **LU** decomposition of the matrix \mathbf{A}
2. Solve $\mathbf{A}\mathbf{u} = \mathbf{b}$
3. Form $\dot{\mathbf{b}}_{new} = \dot{\mathbf{b}} - \dot{\mathbf{A}}\mathbf{u}$
4. Solve $\mathbf{A}\dot{\mathbf{u}} = \dot{\mathbf{b}}_{new}$

This speeds up the differentiated code by saving the cost of a **LU** decomposition. In the reverse mode, the following procedure [11] is used:

- | | |
|---|--|
| <ol style="list-style-type: none"> (a). In the forward sweep 1. Perform an LU decomposition of the matrix \mathbf{A} 2. Store \mathbf{L} and \mathbf{U} and the pivot sequence IPIV 3. Solve $\mathbf{A}\mathbf{u} = \mathbf{b}$ | <ol style="list-style-type: none"> (b). In the reverse sweep 1. Load \mathbf{L} and \mathbf{U} and the pivot sequence IPIV 2. Solve $\mathbf{A}^T\bar{\mathbf{b}} = \bar{\mathbf{u}}$ 3. Update $\bar{\mathbf{A}} = \bar{\mathbf{A}} - \bar{\mathbf{b}}\bar{\mathbf{b}}^T$ |
|---|--|

Here, the memory storage is dramatically reduced. If \mathbf{A} is a $n \times n$ matrix, we stored only n^2 complex coefficients instead of n^3 . We also gain by not reversing the **LU** decomposition procedure.

2.4 Initial Results and Validation

After implementing the procedures described in Subsection 2.2 and 2.3 on the ADIFOR 3.0 generated code, we obtained derivative codes that calculate directional derivatives consistent with FD. The obtained codes were compiled with maximum optimisations and run on a Sun Blade 1000 machine. Table 1 shows the results and timings of forward mode AD, reverse mode AD, and one-sided FD for that calculation. These results showed that forward and reverse AD gave the same directional derivative value within the relative precision of the computer and roundoff while the maximum difference with FD result is around 10^{-6} . This difference is within the square root of the relative precision of the machine. This validates the AD results as being in agreement with the FD result.

Method	$\langle \bar{\mathbf{x}}, \dot{\mathbf{x}} \rangle$	$\langle \bar{\mathbf{y}}, \dot{\mathbf{y}} \rangle$	CPU($\nabla \mathbf{F}(\mathbf{x})\dot{\mathbf{x}}$)
FD (1-sided)		0.124578003587	48.7
ADIFOR 3.0(fwd)		0.124571139127	54.0
ADIFOR 3.0(rev)	0.124571139130		311.5

Table 1: Results for a single directional derivative, timings are in CPU seconds

From Table 1, we can also deduce that while the AD reverse mode calculates the gradient at around 5 minutes, the one-sided FD and forward AD will respectively require 454 function evaluations 453 directional derivatives and consequently runtime of over 3 and 7 hours. We see that using reverse mode AD can speed up the gradient calculation at a factor of around 35 over FD while giving accurate derivatives. However, the core of the calculation (building up the linear system, solving it and calculating the local energy contribution for each frequency) of the BEAM3D code is an independent loop and therefore can be differentiated in parallel as we now describe.

3 Performance Issues

Usually, after checking that the Automatic Differentiation forward and reverse modes agreed with the finite-differences, we seek to improve efficiency of the automatically generated code. As shown by the results of Table 1, the reverse mode is superior to the finite-differences and forward mode but it requires a very large amount of memory to run. This memory is due to the size of the tape which required 12 GB. By hand-coding the adjoint of the linear solver, we reduced the size of the tape to around 6 GB. Furthermore, because the core of the calculation is done by a parallel loop, we can adjoin the body of the loop after each iteration [3]. This reduced the tape size of the adjoint code down to around 0.3 GB.

The second row of Table 2 shows that after this optimisation of the parallel loop, the ratio between the gradient calculation and the function is 8.2. It also shows a speed up factor of 56.6 of the ADIFOR 3.0 generated by reverse mode followed by hand-coded procedures over the popular one-sided FD method.

Method	CPU($\nabla \mathbf{F}$)	CPU($\nabla \mathbf{F}$)/CPU(\mathbf{F})
ADIFOR 3.0(rev.)	311.5	13.3
ADIFOR 3.0(rev.,par.)	192.0	8.2
FD (1-sided)	10912.7	464.4

Table 2: CPU Timings (in Seconds) on a SUN Blade 1000, UltraSparcIII

4 Conclusions

ADIFOR 3.0 allowed us to build up an adjoint for a code using complex variable arithmetic that accurately, calculates the gradient of a cost function. The obtained adjoint code requires only 8.2 times the CPU time of the original function code and the memory requirement for taping is a modest 0.3 GB. It also runs 56.6 times faster than the gradient calculated using the one-sided finite differences. This reduction in computational time of the gradient calculation will be of great benefit for the genetic algorithm-local search method to be used to optimise the design of the satellite boom.

Acknowledgements

This work is supported by EPSRC under grant GR/R85358/01. The authors would like to thank Mike Fagan for his help in using ADIFOR 3.0.

References

- [1] A. Carle and M. Fagan. ADIFOR 3.0 overview. Technical Report CAAM-TR-00-02, Rice University, 2000.
- [2] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, Penn., 2000.
- [3] L. Hascoët, S. Fidanova, and C. Held. Adjoining independent computations. In G. Corliss, C. Faure, A. Griewank, L. Hascoët, and U. Naumann, editors, *Automatic Differentiation: From Simulation to Optimization*, Computer and Information Science, chapter 35, pages 285–290. Springer, New York, 2001.
- [4] A. Keane and S. Brown. The design of a satellite boom with enhanced vibration performance using genetic algorithm techniques. In *Proceedings of ACEDC'96*, PEDC, University of Plymouth, UK, 1996.
- [5] A. J. Keane. Passive vibration control via unusual geometries: The application of genetic algorithm optimization to structural design. *Journal of Sound and Vibration*, 185(3):441–453, 1995.
- [6] M. Moshfreti-Torbati, A. Keane, S. E. M. Brennan, and E. Rogers. The integration of advanced active and passive structural vibration control. In *Proceedings of VETOMAC-I*, Bangalore, India, October 25-27 2000.
- [7] Y. Ong and A. Keane. Meta-lamarckian learning in memetic algorithms. *IEEE Trans. Evolutionary Computing*, 8(2), 2004.
- [8] G. D. Pusch, C. Bischof, and A. Carle. On automatic differentiation of codes with complex arithmetic with respect to real variables. Technical Report ANL/MCS-TM-188, Mathematics and Computer Science Division, Argonne National Laboratory, June 1995.
- [9] K. Shankar and A. J. Keane. Energy flow predictions in a structure of rigidly joined beams using receptance theory. *Journal of Sound and Vibration*, 185(5):867–890, 1995.
- [10] K. Shankar and A. J. Keane. A study of the vibrational energies of two coupled beams by finite element and green function (receptance) methods. *Journal of Sound and Vibration*, 181(5):801–838, 1995.
- [11] A. Verma. *Structured Automatic Differentiation*. PhD thesis, Cornell University Department of Computer Science, Ithaca, NY, 1998.