Automatic Differentiation: Applications, Theory, and Implementations,
Bücker, M.; Corliss, G.; Hovland, P.; Naumann, U.; Norris, B. (Eds.)
Lecture Notes in Computational Science & Engineering, Volume 50, p309-319, Springer, 2006.
ISBN 3-540-28403-6

1

Adjoint Differentiation of a Structural Dynamics Solver*

Mohamed Tadjouddine[1], Shaun A. Forth[1], and Andy J. Keane[2]

[1] Applied Mathematics & Operational Research, ESD
Cranfield University (RMCS Shrivenham), Swindon SN6 8LA, UK
{M.Tadjouddine, S.A.Forth}@cranfield.ac.uk
[2] Computational Engineering and Design Group, School of Engineering Sciences,
University of Southampton, Southampton SO17 1BJ, UK
andy.keane@soton.ac.uk

Summary. 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 7.4 times the time of the function evaluation. This should reduce the rather time-consuming process (over 10 CPU days by using parallel processing) of the GA optimiser for this problem.

Key words: Reverse mode AD, hybrid GA-local search, structural dynamics, performance.

1.1 Introduction

In space missions, lightweight cantilever structures are often used to suspend scientific instruments, such as antenna, a few metres away from the satellite. An example of this kind of structure is the satellite boom shown in Fig. 1.1. Vibrations can be transmitted through the structure from satellite to the instrument. Such vibrations can damage the boom structure or prevent it from being used for its intended purpose. To ensure correct functioning of the instrument, the vibrations or mechanical noise through the structure must be kept at tolerable levels. Typically, the structure is excited by a point transverse force near an end beam, and the energy level is measured at the opposite end

*This work is supported by UK's EPSRC under grant GR/R85358/01

beam. To minimise vibrations and noise, the geometry of the structure is modified to reduce the frequency average response of the satellite boom. This is known as *passive vibration control* [1].

In this paper, we consider the satellite boom of Fig. 1.1 described in Sect. 1.2.1 and previously studied in [2, 3]. The structural dynamics of the

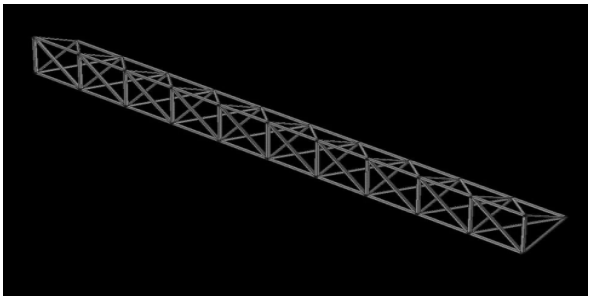

Fig. 1.1. Initial geometry of the satellite boom

three-dimensional satellite boom are modelled by a Fortran computer code named BEAM3D [4, 5] using 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 shapes.

In previous work [2, 3], the minimisation of the frequency average response (in the range 150–250 Hz) at the end beam was carried out to find a superior design or geometry. For that purpose, a Genetic Algorithm (GA) was used. GAs are known to work for fairly large cost with a good chance of finding 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 [2].

Attention has now shifted to a hybrid genetic algorithm-local search approach combining Darwinian and Lamarckian evolution models. Darwinian evolution, based on "natural selection" considers that the most fit individuals are likely to survive. Lamarckian evolution takes the view that individuals may improve within their environment as they become adapted to it and that the resulting changes are inheritable. Consequently, the genotype of an improved individual is forced to reflect the result of such an improvement by replacing the individual into the population for reproduction [6]. In our application, the local search is carried out using a gradient method, which requires the calculation of the gradient of a cost function $F : \mathbb{R}^{90} \rightarrow \mathbb{R}$. Here, the 90 inputs are the coordinates of the joint positions in the design geometry and represent the independent variables. The computation of transmitted power $F(\mathbf{x})$ involves complex variable calculations, which are handled by ADIFOR 3.0 [7] since the dependents and independents are real values [8].

In theory, the gradient ∇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 and is bounded above by a small factor of the cost of evaluating the function [9]. In practice, large memory requirement may prohibit use of the adjoint code (reverse AD generated code). This paper details the differentiation of the BEAM3D code by the ADIFOR 3.0, the successor of the AD tool ADIFOR 2.0 [10]. ADIFOR 3.0 employed in reverse mode produces an adjoint code which, after being tuned manually for performance enhancement, calculated the function and its gradient in 7.4 times the CPU time required for its function evaluation. Moreover, the adjoint code runs 12.6 times faster than one-sided finite-differencing (FD) on a Sun Blade 1000 machine with 1200 MHz CPU, 8 MB external cache and 2 GB RAM.

1.2 Optimisation of the Boom Structure

We aim at minimising vibrations through the structure represented in Fig. 1.1. There are at least three ways to achieve this: increasing the mass of elements or coating elements with damping material, using active anti-vibration to cancel unwanted vibrations; and as considered here, modifying the geometry of the structure to filter and reflect the vibrations.

1.2.1 The Initial Geometry

The initial boom structure to be optimised is three-dimensional and composed of 90 Euler-Bernouilli beams each having the same properties per unit length. Because the structure is used to mount a scientific instrument away from a space satellite, the length of the boom structure must be chosen within reasonable limits. Typical values of the aluminium were used for the physical properties of the beams. The bay length is 45 cm, and the overall length of the boom structure is 4.5 m.

The beams were arranged in a regular manner along the XYZ axes so that the YZ cross-section of the boom structure formed an equilateral triangle. The three joints at the left hand end of the structure were fixed, i.e., they were clamped to prevent motion. The beams were connected together with 30 free joints. Geometric constraints were used to avoid beams overlapping or becoming extremely long. The free joints were kept within fixed distances of their original positions. The connectivity of the diagonal beams was chosen so that a maximum of six beams met at any one joint.

Typically, the structure is excited by a point transverse force applied to a left hand end beam of the structure. The vibrational energy level is calculated at a right hand end beam using receptance methods [1]. The optimisation aims at minimising the vibrations by minimising the frequency averaged response in the range $150-250$ Hz. For that purpose, the optimiser is allowed to modify the geometry of the satellite boom by changing the coordinates of the 30 free joints in the structure.

1.2.2 An Optimised Geometry

A GA from the optimisation software package [11] was used to generate an optimised boom geometry by improving the frequency response curve. The principles of a GA can be found in [12]. In short, GAs work on the premise that a population of competing individuals can be combined to produce improved individuals. They basically mimic "natural" selection, or Darwinian evolution. Common operations are:

- selection: whereby the fittest individuals are chosen to "inter-breed" and pass their attributes to their offspring.
- crossover: where random portions of two of the most fit individuals are combined to form a new individual.
- mutation: where small changes are introduced to one individual at a time.

Furthermore, the number of generations and their population size are usually chosen in advance.

As reported in [3], an optimised design geometry was obtained using an objective function F set to be the square root of the sum of velocity squared in the X, Y, Z directions for the end three joints labelled $31, 32$ and 33,

$$F = \int_{\text{freq}} \sum_{j=31}^{33} \sqrt{V_{xj}^2 + V_{yj}^2 + V_{zj}^2}.$$

A run of the GA for 10 generations and a population size of 300, gave the novel design geometry of the boom structure shown in Fig. 1.2. However, GAs applied to large-scale optimisation problems can take CPU days even using parallel processing [2]. To enhance their performance, they may be combined with local search methods.

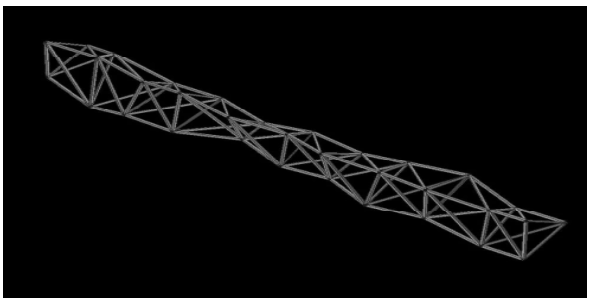

Fig. 1.2. An optimised geometry of the satellite boom

1.2.3 Coupling GAs and Local Search Methods

For the optimisation process of the satellite boom of Fig. 1.1, we believe a GA coupled with a local search method should outperform a stand-alone GA.

Such local search methods are based on gradient descent. We aim to make them efficient by taking advantage of the accuracy and efficiency of gradient calculation by reverse mode AD.

In essence, the hybridised GA-Local Search method performs similar steps to that of the GA except that each individual of the population is locally improved using a local search method, here steepest descents, following the meta-Lamarkian learning approach as shown in Fig. 1.3 and detailed in [6]. In Fig. 1.3, the while loop is executed until either t exceeds some maximum number of generations t_{max} or convergence is detected. We now detail the differentiation of the BEAM3D code to enable the gradient descent method of the hybrid algorithm.

$t = 0$
Initialise a GA population $P(t) = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m\}$
While (EndCondition is not satisfied)
  Evaluate Fitness$(P(t))$ giving $F(\mathbf{x}^1), F(\mathbf{x}^2), \ldots, F(\mathbf{x}^m)$
  For each individual $\mathbf{x}^i \in P(t)$
    Improve $\mathbf{x}^i$ using the Gradient Descent method GD$(\mathbf{x}^i)$
    Replace $\mathbf{x}^i$ by the improved $\mathbf{x}^i_{new} =$ GD$(\mathbf{x}^i)$ in $P(t)$
  EndFor
  Generate $P(t+1)$ from the $\mathbf{x}^i_{new}$ by using standard
  GA operations (Selection, Mutation, or Crossover).
  $t = t + 1$
EndDo

Fig. 1.3. Hybrid GA-Gradient Descent Method

1.3 Differentiation of the BEAM3D Code

The BEAM3D code, as sketched in Fig. 1.4, starts by reading in data from files representing the boom geometry and certain properties of each beam. Given extra information such as: the range of frequencies over which to solve, the number of data points within the specified frequency range, the joint numbers at which to calculate the energies and the number of axial[torsional] and transverse modes in the modal summations for the Green functions; the program builds up a linear complex system $\mathbf{Af} = \mathbf{b}$ for nodal forces $\mathbf{f}$ and solves it for each frequency. An averaged energy function is calculated at the specified end beam.

We aim to calculate the sensitivities of the energy function with respect to the coordinates of the free joints. This represents a gradient calculation with 90 independent variables. Prior to differentiation, the code was restructured so all reading of data is done outside the subroutines to be differentiated. Furthermore, to allow the code to be processed by ADIFOR 3.0, the code was rewritten according to the Fortran 77 standard. Actually, the original code contained (non-standard) language extensions in the form of structures defined as follows:

```
Read in:
     No. of beams, beam properties, and list of connections
     Frequency range [ω_min, ω_max, N] (N ≡ No. of frequencies)
     Coordinates of beam ends x_{n,j}
Calculate some geometric information
Initialise F = 0 (integral of power)
Δω = (ω_max − ω_min)/(N − 1)
For k = 1, N
     ω = ω_min + (k − 1) ∗ Δω
     Assemble:
          Green Function Matrix A(x, iω)
          r.h.s. forcing b(x, iω)
     Solve A(x, iω)f = b(x, iω) (LAPACK) - 50% of CPU time
     Obtain displacement D_{n,j} at ends of beam n
     Obtain power P = (1/2) Re(f_j · D_{n,j})
     Update integral F = F + P^2 ∗ Δω
End For
```

Fig. 1.4. Schematic of the BEAM3D code

```
STRUCTURE /PROPERTY/
          INTEGER ID
          CHARACTER*6 ENDCON
          DOUBLE PRECISION ANGLE(3,3)
          DOUBLE PRECISION LENG
          ...
          COMPLEX *16  FM2
 END STRUCTURE
 RECORD /PROPERTY/ BEAM(150)
```

This structure is replaced using arrays corresponding to the components of the structure. The restructured code contains the following array declarations:

```
INTEGER BEAM_ID(150)
CHARACTER*6 BEAM_ENDCON(150)
DOUBLE PRECISION BEAM_ANGLE(3,3,150)
DOUBLE PRECISION BEAM_LENG(150)
...
COMPLEX*16 BEAM_FM2(150)
```

A `sed` [13] script was written to replace any instance `BEAM(I).X(K,J)`, where `X` represents any component of the structure, by the array element `BEAM_X(K,J,I)`. If `X` is a scalar variable, obviously no indices are used. The resulting computer code is differentiated using FD, and AD via ADIFOR 3.0.

1.3.1 Initial Differentiation

We first computed a single directional derivative $\dot{\mathbf{y}} = \nabla F(\mathbf{x})\dot{\mathbf{x}}$ for a random direction $\dot{\mathbf{x}}$, by using one-sided FD, AD in forward mode, and a single adjoint $\bar{\mathbf{x}} = \nabla F(\mathbf{x})^T \bar{\mathbf{y}}$ for $\bar{\mathbf{y}} = 1$ via reverse mode AD. By definition of the adjoint operator, we have the following equality: $\bar{\mathbf{y}}\dot{\mathbf{y}} \equiv \bar{\mathbf{x}}\dot{\mathbf{x}}$. This allows us to validate the results of the differentiation. The initial ADIFOR 3.0 generated codes gave incorrect results being inconsistent with those from FD. This is caused partly by non-differentiable statements in the code for the function F.

1.3.2 Dealing With Non-differentiability

A major assumption in Automatic Differentiation is that the function F to be differentiated is composed of elemental functions ϕ that are continuously differentiable on their open domains [9]. At a point on the boundary of an open domain, F is continuous, but ∇F may jump to a finite value or even infinity. This becomes an important 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`). To compute reliable derivatives, such pathological cases must be handled correctly. It is known that these cases can be tackled by calculating derivatives in a given direction [9]. Insights or knowledge of the computer code can also be exploited. The BEAM3D code contains at least two types of non-differentiability.

```
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))
```

Fig. 1.5. A code fragment that is non-differentiable

The first type of non-differentiability is due to the presence of the functions `arccos` and `abs`. ADIFOR 3.0 allows us to locate possible non-differentiable points by generating the derivative code with the *Exception Handling* enabled [7]. On running this code, warnings were raised concerning the functions `arccos` and `abs`. The part of the code, which caused such anomalies is shown in Fig. 1.5.

We then used algebra and trigonometric formula to rewrite some of the algebraic expressions containing (`arccos`, `abs`, `sin` and `tan`) as in Fig. 1.6.

```
myt1 = ydiff/beam_leng(i)
myt2 = dsqrt(xdiff*xdiff+zdiff*zdiff)
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*myt1*xdiff/myt2
yor(2,i) = sgn2*myt2/beam_leng(i)
yor(3,i) = sgn3*myt1*zdiff/myt2
```

Fig. 1.6. An equivalent but differentiable version of the code fragment of Fig. 1.5

This transformation resulted in equivalent expressions calculating the same values but differentiable in the vicinity of their arguments.

The second type of non-differentiability encountered in BEAM3D was due to a branching construct illustrated by the code fragment of Fig. 1.7. As `xdiff` and `ydiff` are active variables, the differentiation of this code fragment gave point-valued derivatives that prevented the function **F** from being differentiable. Such branches represent constraints on the design geometry and, in our case, may be safely removed.

```
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 ....
```

Fig. 1.7. A code fragment testing whether an active variable is zero

Finally, the complex linear solver **Af**=**b** employs the LAPACK routine `zgesv` [14]. Differentiating the LAPACK source code routines for `zgesv` using an AD tool without taking account insights into the nature of the linear solver would give inefficient code. Mechanical generation of the `zgesv` derivative by ADIFOR 3.0 gave not only inefficient code but also incorrect results, being inconsistent with FD. We therefore hand-coded its derivative as described in Sect. 1.3.3.

1.3.3 Complex Linear Solver

Instead of using ADIFOR 3.0 to differentiate the complex linear solve,

$$\mathbf{A}\mathbf{f} = \mathbf{b}, \tag{1.1}$$

of the LAPACK routine `zgesv`, we instead use hand-coding for both forward and reverse mode. Differentiating $\mathbf{A}\mathbf{f} = \mathbf{b}$, using the matrix-equivalent of the product rule for a single directional derivative, we obtain $\mathbf{A}\dot{\mathbf{f}} + \dot{\mathbf{A}}\mathbf{f} = \dot{\mathbf{b}}$, and so the derivatives $\dot{\mathbf{f}}$ are given by the solution of

$$\mathbf{A}\dot{\mathbf{f}} = \dot{\mathbf{b}} - \dot{\mathbf{A}}\mathbf{f}. \tag{1.2}$$

In the forward mode we may re-use the LU-decomposition of $\mathbf{A}$ to efficiently solve for the derivatives $\dot{\mathbf{f}}$. The following procedure is used:

1. Perform an $\mathbf{LU}$ decomposition of the matrix $\mathbf{A}$
2. Solve $\mathbf{A}\mathbf{f} = \mathbf{b}$
3. Form $\mathbf{b}_{new} = \dot{\mathbf{b}} - \dot{\mathbf{A}}\mathbf{f}$
4. Re-use LU-decomposition to solve $\mathbf{A}\dot{\mathbf{f}} = \mathbf{b}_{new}$

Of course $\dot{\mathbf{A}}$ and $\dot{\mathbf{b}}$ are calculated by applying ADIFOR 3.0 to the `Assemble` procedure in Fig. 1.4.

For the reverse mode, deriving the adjoint update corresponding to (1.1) is more problematic. Defining $\mathbf{C} = \mathbf{A}^{-1}$ then we may write,

$$\dot{\mathbf{f}} = \mathbf{C}\dot{\mathbf{b}} - \mathbf{C}\dot{\mathbf{A}}\mathbf{f}.$$

Then $\dot{f}_i$, the i^{th} element of $\dot{\mathbf{f}}$, is given by

$$\dot{f}_i = \sum_j c_{ij}\dot{b}_j - \sum_j c_{ij} \sum_k \dot{a}_{jk} f_k, \tag{1.3}$$

where c_{ij}, $\dot{a}_{jk}$, $\dot{b}_j$ and f_k are the elements of $\mathbf{C}$, $\dot{\mathbf{A}}$, $\dot{\mathbf{b}}$ and $\mathbf{f}$ respectively. Now we use the identity $\bar{\mathbf{y}}\dot{\mathbf{y}} \equiv \bar{\mathbf{x}}\dot{\mathbf{x}}$ for the system $\mathbf{y} = \mathbf{F}(\mathbf{x})$ [9, Equation (3.7)]. In the context of the vector and matrix arguments of the system (1.1), this identity gives

$$\sum_i \bar{f}_i \dot{f}_i = \sum_i \bar{b}_i \dot{b}_i + \sum_i \sum_j \bar{a}_{ij} \dot{a}_{ij}. \tag{1.4}$$

From (1.3) we obtain

$$\sum_i \bar{f}_i \dot{f}_i = \sum_i \bar{f}_i \sum_j c_{ij}\dot{b}_j - \sum_i \bar{f}_i \sum_j c_{ij} \sum_k \dot{a}_{jk} f_k,$$

by reordering summations,

$$\sum_i \bar{f}_i \dot{f}_i = \sum_j \dot{b}_j \sum_i c_{ij}\bar{f}_i - \sum_j \sum_k \dot{a}_{jk} f_k \sum_i c_{ij}\bar{f}_i,$$

and by swapping indices (i, j, k) to (k, i, j),

$$\sum_i \bar{f}_i \dot{f}_i = \sum_i \dot{b}_i \sum_j c_{ji}\bar{f}_j - \sum_i \sum_j \dot{a}_{ij} f_j \sum_k c_{ki}\bar{f}_k.$$

Comparing with (1.4) we see that

$$\bar{b}_i = \sum_j c_{ji} \bar{f}_j,$$

giving $\bar{\mathbf{b}} = \mathbf{C}^T \bar{\mathbf{f}} = \mathbf{A}^{-T} \bar{\mathbf{f}}$ or that $\bar{\mathbf{b}}$ is given as the solution of

$$\mathbf{A}^T \bar{\mathbf{b}} = \bar{\mathbf{f}}. \tag{1.5}$$

Similarly,

$$\bar{a}_{ij} = -f_j \sum_k c_{ki} \bar{f}_k = -f_j \bar{b}_i,$$

or

$$\bar{\mathbf{A}} = -\bar{\mathbf{b}} \mathbf{f}^T. \tag{1.6}$$

The adjoint $\bar{\mathbf{b}}$ is updated by solving the linear system (1.5), while $\bar{\mathbf{A}}$ is updated by adding the right hand side of the equation (1.6). Adjoint formulae (1.5) and (1.6) are equivalent to those given in [15].

Using (1.5) and (1.6) we obtain the following procedure for the adjoint of the linear solve:

1. In the forward sweep,
   a) Perform an **LU** decomposition of the matrix **A**,
   b) Store **L** and **U** and the pivot sequence `IPIV`,
   c) Solve $\mathbf{A}\mathbf{f} = \mathbf{b}$.
2. In the reverse sweep,
   a) Load **L** and **U** and the pivot sequence `IPIV`,
   b) Solve $\mathbf{A}^T \bar{\mathbf{b}} = \bar{\mathbf{f}}$ for $\bar{\mathbf{b}}$,
   c) Update $\bar{\mathbf{A}} = \bar{\mathbf{A}} - \bar{\mathbf{b}} \mathbf{f}^T$.

Note that since both **A** and **b** depend on the beam endpoint location **x** (see Fig. 1.4), the adjoined linear solver must modify their adjoints $\bar{\mathbf{A}}$ and $\bar{\mathbf{b}}$. The adjoint for $\bar{\mathbf{A}}$ is the usual increment, while for $\bar{\mathbf{b}}$ it is an assignment because the LAPACK routine `zgesv` overwrites **b** with **f**. Here, the memory storage is dramatically reduced compared with black-box application of ADIFOR 3.0. If **A** is an $N \times N$ matrix, we store only N^2 complex coefficients instead of $O(N^3)$ when ADIFOR 3.0 tapes all variables on the left of assignment statements in the LU decomposition.

1.3.4 Initial Results and Validation

After implementing the procedures described in Sect. 1.3.2 and 1.3.3 on the ADIFOR 3.0 generated code, we obtained tangent and adjoint derivative codes that calculate directional derivatives consistent with one-sided FD. The obtained codes were compiled with maximum compile optimisations and run on a Sun Blade 1000 machine. Table 1.1 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 roundoff, while the maximum difference with the FD result is around 10^{-6}. This difference is of the order of the square root of the machine relative precision. This validates the AD results as being in agreement with the one-sided FD result.

Table 1.1. Results for a single directional derivative, timings are in CPU seconds

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

From Table 1.1, we can also deduce that while the AD reverse mode calculates the gradient in around 5 minutes, one-sided FD and forward AD requires 91 function evaluations and 90 directional derivatives respectively and consequently run-times of over 35 minutes. We see that using reverse mode AD can speed up the gradient calculation by a factor of around 7 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.

1.4 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 1.1, the reverse mode is superior to the finite-differences and forward mode, but we found it requires a very large amount of memory to run because the tape required 12 GB. By hand-coding the adjoint of the linear solver, we reduced the size of the tape to around 6 GB.

Furthermore, the core of the calculation of the BEAM3D code is carried out in a parallel loop, which is the loop over k in Fig. 1.4. Because the iterations of such a loop are independent, we can run the loop body taping all the required information in just one iteration, then immediately adjoin the body of the loop [16]. This reduced the tape size of the adjoint code to around 0.3 GB. This represents a memory reduction by a factor of 20, which is the number of extra iterations performed by the parallel loop.

The second row of Table 1.2 shows that after this optimisation, the ratio between the gradient calculation and the function is 7.4. It also shows a speed up factor of 12.6 over the popular one-sided FD method.

Table 1.2. CPU Timings (in Seconds) on a SUN Blade 1000, UltraSparcIII

Method	CPU$(F, \nabla F)$	CPU$(F, \nabla F)$/CPU(F)
ADIFOR 3.0(rev.)	311.5	13.3
ADIFOR 3.0(rev.,par.)	174.7	7.4
FD (1-sided)	2215.9	93.1

1.5 Conclusions

ADIFOR 3.0 allowed us to build an adjoint for a code that makes extensive use of complex variable arithmetic to accurately calculate the gradient of a cost function. The adjoint code requires only 7.4 times the CPU time of the original function code, and the memory requirement for taping is a modest 0.3 GB. It also runs 12.6 times faster than calculating the gradient using one-sided finite differencing.

Future work is planned to compare the performance of gradient calculation using both ADIFOR 3.0 [7] and TAF [17] capabilities. The design optimisation of the lightweight cantilever structure will be carried out using the meta-Lamarckian learning strategy [6], which efficiently combines GAs with local search methods. The reduction in computational time of the gradient calculation will be of great benefit in allowing the meta-Lamarckian algorithm to be used to optimise the design of boom structures.

Acknowledgements

The authors thank the UK's EPSRC for funding this project under grant GR/R85358/01 and Mike Fagan for his help in using ADIFOR 3.0.

References

1. Keane, A.J.: Passive vibration control via unusual geometries: The application of genetic algorithm optimization to structural design. Journal of Sound and Vibration **185** (1995) 441–453
2. Keane, A., Brown, S.: 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)
3. Moshfreti-Torbati, M., Keane, A., Brennan, S.E.M., Rogers, E.: The integration of advanced active and passive structural vibration control. In: Proceedings of VETOMAC-I, Banglore, India (2000)
4. Shankar, K., Keane, A.J.: Energy flow predictions in a structure of rigidly joined beams using receptance theory. Journal of Sound and Vibration **185** (1995) 867–890
5. Shankar, K., Keane, A.J.: A study of the vibrational energies of two coupled beams by finite element and Green function (receptance) methods. Journal of Sound and Vibration **181** (1995) 801–838
6. Ong, Y., Keane, A.: Meta-Lamarckian learning in memetic algorithms. IEEE Trans. Evolutionary Computing **8** (2004)
7. Carle, A., Fagan, M.: ADIFOR 3.0 overview. Technical Report CAAM-TR-00-02, Rice University (2000)
8. Pusch, G.D., Bischof, C., Carle, A.: 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 (1995)
9. Griewank, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, Penn. (2000)
10. Bischof, C.H., Carle, A., Khademi, P., Mauer, A.: ADIFOR 2.0: Automatic differentiation of Fortran 77 programs. IEEE Computational Science & Engineering **3** (1996) 18–32
11. Keane, A.: The OPTIONS design exploration system user guide and reference manual. Technical report, Computational Engineering and Design Group, School of Engineering Sciences, University of Southampton, Southampton SO17 1BJ, UK (2001) See `http://www.soton.ac.uk/~ajk/options/welcome.html`.
12. Goldberg, D.: Genetic Algorithms in Search, Optimisation and Machine Learning. Adison-Wesley (1989)
13. Dougherty, D.: Sed & awk. O'Reilly & Associates, Iinc. (1992)
14. Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., Sorensen, D.: LAPACK User's Guide, 2nd edn. SIAM, Philadelphia, Penn. (1995)
15. Verma, A.: Structured Automatic Differentiation. PhD thesis, Cornell University Department of Computer Science, Ithaca, NY (1998)
16. Hascoët, L., Fidanova, S., Held, C.: Adjoining independent computations. In Corliss, G., Faure, C., Griewank, A., Hascoët, L., Naumann, U., eds.: Automatic Differentiation: From Simulation to Optimization. Computer and Information Science. Springer, New York (2001) 285–290
17. FastOpt: Transformation of Algorithms in Fortran, Manual, Draft Version, TAF Version 1.6. (2003) See `http://www.FastOpt.com/taf`.