Activity Report 2011

Project-Team TROPICS

Program transformations for scientific computing

RESEARCH CENTER
Sophia Antipolis - Méditerranée

THEME
Computational models and simulation

Table of contents

1. Members **1**
2. Overall Objectives **1**
3. Scientific Foundations **2**
3.1. Automatic Differentiation 2
3.2. Static Analysis and Transformation of programs 3
3.3. Automatic Differentiation and Scientific Computing 4
4. Application Domains **5**
4.1. Panorama 5
4.2. Multidisciplinary optimization 5
4.3. Inverse problems and Data Assimilation 5
4.4. Linearization 6
4.5. Mesh adaptation 7
5. Software **7**
5.1. AIRONUM 7
5.2. TAPENADE 7
6. New Results **8**
6.1. Automatic Differentiation and parallel codes 8
6.2. AD adjoints and Dynamic Memory 9
6.3. Resolution of linearised systems 9
6.4. Perturbation Methods 10
6.5. Control of approximation errors 10
7. Partnerships and Cooperations **11**
8. Dissemination **11**
8.1. Animation of the scientific community 11
8.2. Teaching 12
9. Bibliography **12**

Project-Team TROPICS

Keywords: Scientific Computation, Fluid Dynamics, Automatic Differentiation, Program Transformation, Parallelism

1. Members

Research Scientists

Laurent Hascoët [Senior Researcher, Team leader, HdR]
Valérie Pascual [Junior Researcher]
Alain Dervieux [Senior Researcher, HdR]

Faculty Member

Bruno Koobus [Université Montpellier 2, HdR]

External Collaborators

Stephen Wornom [Lemma Company]
Trond Steihaug [Professor, University of Bergen, Norway, on sabbatical]

PhD Students

Anca Belme
Hubert Alcin
Alexandre Carabias

Administrative Assistant

Claire Senica

2. Overall Objectives

2.1. Overall Objectives

The TROPICS team studies Automatic Differentiation (AD) of algorithms and programs. We work at the junction of two research domains:

- **AD theory:** On the one hand, we study software engineering techniques, to analyze and transform programs mechanically. Automatic Differentiation (AD) transforms a program `P` that computes a function F, into a program `P'` that computes analytical derivatives of F. We put emphasis on the so-called *reverse* or *adjoint* mode of AD, a sophisticated transformation that yields gradients for optimization at a remarkably low cost.
- **AD application to Scientific Computing:** On the other hand, we study application of the adjoint mode of AD to e.g. Computational Fluid Dynamics. We adapt the strategies used in Scientific Computing in order to take full advantage of AD. This work is applied to several real-size applications.

Each aspect of our work benefit to the other. We want to produce AD code that can compete with hand-written sensitivity and adjoint programs that are used in the industry. We implement our algorithms into our tool TAPENADE, which is now one of the most popular AD tools.

Our research directions are :

- Modern numerical methods for finite elements or finite differences : multigrid methods, mesh adaptation.
- Optimal shape design or optimal control in fluid dynamics for steady and unsteady simulations. Higher-order derivatives needed by robust optimization.
- Automatic Differentiation : AD-specific static data-flow analysis, strategies to reduce runtime and memory consumption of the reverse mode in the case of very large codes. Improved models for reverse AD, in particular coping with message-passing parallellism.

3. Scientific Foundations

3.1. Automatic Differentiation

Participants: Laurent Hascoët, Valérie Pascual.

Glossary

automatic differentiation (AD) Automatic transformation of a program, that returns a new program that computes some derivatives of the given initial program, i.e. some combination of the partial derivatives of the program's outputs with respect to its inputs.

adjoint model Mathematical manipulation of the Partial Derivative Equations that define a problem, obtaining new differential equations that define the gradient of the original problem's solution.

checkpointing General trade-off technique, used in the reverse mode of AD, that trades duplicate execution of a part of the program to save some memory space that was used to save intermediate results. Checkpointing a code fragment amounts to running this fragment without any storage of intermediate values, thus saving memory space. Later, when such an intermediate value is required, the fragment is run a second time to obtain the required values.

Automatic or Algorithmic Differentiation (AD) differentiates *programs*. An AD tool takes as input a source computer program P that, given a vector argument $X \in I\!R^n$, computes some vector function $Y = F(X) \in I\!R^m$. The AD tool generates a new source program P' that, given the argument X, computes some derivatives of F. The resulting P' reuses the control of P.

For any given control, P is equivalent to a sequence of instructions, which is identified with a composition of vector functions. Thus, if

$$\begin{array}{lcl} P & \text{is} & \{I_1; I_2; \cdots I_p;\}, \\ F & = & f_p \circ f_{p-1} \circ \cdots \circ f_1, \end{array} \tag{1}$$

where each f_k is the elementary function implemented by instruction I_k. AD applies the chain rule to obtain derivatives of F. Calling X_k the values of all variables after instruction I_k, i.e. $X_0 = X$ and $X_k = f_k(X_{k-1})$, the chain rule gives the Jacobian of F

$$F'(X) = f'_p(X_{p-1}) \,.\, f'_{p-1}(X_{p-2}) \,.\, \cdots \,.\, f'_1(X_0) \tag{2}$$

which can be mechanically written as a sequence of instructions I'_k. Combining the I'_k with the control of P yields P'. This can be generalized to higher level derivatives, Taylor series, etc.

In practice, the Jacobian $F'(X)$ is often too expensive to compute and store, but most applications only need projections of $F'(X)$ such as:

- **Sensitivities**, defined for a given direction $\dot{X}$ in the input space as:

$$F'(X).\dot{X} = f'_p(X_{p-1}) \,.\, f'_{p-1}(X_{p-2}) \,.\, \cdots \,.\, f'_1(X_0) \,.\, \dot{X} \quad . \tag{3}$$

  Sensitivities are easily computed from right to left, interleaved with the original program instructions. This is the *tangent mode* of AD.

- **Adjoints**, defined for a given weighting $\overline{Y}$ of the outputs as:

$$F'^*(X).\overline{Y} = f'^*_1(X_0).f'^*_2(X_1).\, \cdots \,.f'^*_{p-1}(X_{p-2}).f'^*_p(X_{p-1}).\overline{Y} \quad . \tag{4}$$

Adjoints are most efficiently computed from right to left, because matrix×vector products are cheaper than matrix×matrix products. This is the *reverse mode* of AD, most effectiev for optimization, data assimilation [28], adjoint problems [23], or inverse problems.

The reverse mode turns out to make a very efficient program, at least theoretically [25]. The computation time required for the gradient is only a small multiple of the run-time of P. It is independent from the number of parameters n. In contrast, computing the same gradient with the *tangent mode* would require running the tangent differentiated program n times.

However, we observe that the X_k are required in the *inverse* of their computation order. If the original program *overwrites* a part of X_k, the differentiated program must restore X_k before it is used by $f'^*_{k+1}(X_k)$. Therefore, the central research problem of the reverse mode is to make the X_k available in reverse order at the cheapest cost, using strategies that combine storage, repeated forward computation from available previous values, or even inverted computation from available later values.

Another research issue is to make the AD model cope with the constant evolution of modern language constructs. From the old days of Fortran77, novelties include pointers and dynamic allocation, modularity, structured data types, objects, vectorial notation and parallel communication. We regularly extend our models and tools to handle these new constructs.

3.2. Static Analysis and Transformation of programs

Participants: Laurent Hascoët, Valérie Pascual.

Glossary

abstract syntax tree Tree representation of a computer program, that keeps only the semantically significant information and abstracts away syntactic sugar such as indentation, parentheses, or separators.

control flow graph Representation of a procedure body as a directed graph, whose nodes, known as basic blocks, contain each a list of instructions to be executed in sequence, and whose arcs represent all possible control jumps that can occur at run-time.

abstract interpretation Model that describes program static analysis as a special sort of execution, in which all branches of control switches are taken simultaneously, and where computed values are replaced by abstract values from a given *semantic domain*. Each particular analysis gives birth to a specific semantic domain.

data flow analysis Program analysis that studies how a given property of variables evolves with execution of the program. Data Flow analysis is static, therefore studying all possible run-time behaviors and making conservative approximations. A typical data-flow analysis is to detect whether a variable is initialized or not, at any location in the source program.

data dependence analysis Program analysis that studies the itinerary of values during program execution, from the place where a value is generated to the places where it is used, and finally to the place where it is overwritten. The collection of all these itineraries is often stored as a *data dependence graph*, and data flow analysis most often rely on this graph.

data dependence graph Directed graph that relates accesses to program variables, from the write access that defines a new value to the read accesses that use this value, and conversely from the read accesses to the write access that overwrites this value. Dependences express a partial order between operations, that must be preserved to preserve the program's result.

The most obvious example of a program transformation tool is certainly a compiler. Other examples are program translators, that go from one language or formalism to another, or optimizers, that transform a program to make it run better. AD is just one such transformation. These tools use sophisticated analysis [16] to improve the quality of the produced code. These tools share their technological basis. More importantly, there are common mathematical models to specify and analyze them.

An important principle is *abstraction*: the core of a compiler should not bother about syntactic details of the compiled program. The optimization and code generation phases must be independent from the particular input programming language. This is generally achieved using language-specific *front-ends* and *back-ends*. But one can go further: as abstraction goes on, the internal representation becomes more language independent, and semantic constructs can be unified. Analysis can then concentrate on the semantics of a small set of constructs. We advocate an internal representation composed of three levels.

- At the top level is the *call graph*, whose nodes are modules and procedures. Arrows relate nodes that call or import one another. Recursion leads to cycles.
- At the middle level is the *flow graph*, one per procedure. It captures the control flow between atomic instructions.
- At the lowest level are abstract *syntax trees* for the individual atomic instructions. Semantic transformations can benefit from the representation of expressions as directed acyclic graphs, sharing common sub-expressions.

At each level are associated symbol tables, that are nested to capture the notion of visibility scope.

Static program analysis can be defined on this internal representation, which is largely language independent. The simplest analyses on trees can be specified with inference rules [18], [26], [17]. But many analyses are more complex, and better defined on graphs than on trees. This is the case for *data-flow analyses*, that look for run-time properties of variables. Since flow graphs are cyclic, these global analyses generally require an iterative resolution. *Data flow equations* is a practical formalism to describe data-flow analyses. Another formalism is described in [19], which is more precise because it can distinguish separate *instances* of instructions. However it is still based on trees, and its cost forbids application to large codes. *Abstract Interpretation* [20] is a theoretical framework to study complexity and termination of these analyses.

Data flow analyses must be carefully designed to avoid or control combinatorial explosion. At the call graph level, they can run bottom-up or top-down, and they yield more accurate results when they take into account the different call sites of each procedure, which is called *context sensitivity*. At the flow graph level, they can run forwards or backwards, and yield more accurate results when they take into account only the possible execution flows resulting from possible control, which is called *flow sensitivity*.

Even then, data flow analyses are limited, because they are static and thus have very little knowledge of actual run-time values. In addition to the very theoretical limit of *undecidability*, there are practical limitations to how much information one can infer from programs that use arrays [32], [21] or pointers. In general, conservative *over-approximations* are always made that lead to derivative code that is less efficient than possibly achievable.

3.3. Automatic Differentiation and Scientific Computing

Participants: Alain Dervieux, Laurent Hascoët, Bruno Koobus.

Glossary

linearization In Scientific Computing, the mathematical model often consists of Partial Derivative Equations, that are discretized and then solved by a computer program. Linearization of these equations, or alternatively linearization of the computer program, predict the behavior of the model when small perturbations are applied. This is useful when the perturbations are effectively small, as in acoustics, or when one wants the sensitivity of the system with respect to one parameter, as in optimization.

adjoint state Consider a system of Partial Derivative Equations that define some characteristics of a system with respect to some input parameters. Consider one particular scalar characteristic. Its sensitivity, (or gradient) with respect to the input parameters can be defined as the solution of "adjoint" equations, deduced from the original equations through linearization and transposition. The solution of the adjoint equations is known as the adjoint state.

Scientific Computing provides reliable simulations of complex systems. For example it is possible to simulate the 3D air flow around a plane that captures the physical phenomena of shocks and turbulence. Next comes optimization, one degree higher in complexity because it repeatedly simulates and applies optimization steps until an optimum is reached. We focus on gradient-based optimization.

We investigate several approaches to obtain the gradient. There are actually two extreme approaches:

- One can write an *adjoint system* of mathematical equations, then discretize it and program it by hand. This is mathematically sound [23], but very costly in development time. It also does not produce an exact gradient of the discrete function, and this can be a problem if using optimization methods based on descent directions.
- One can apply reverse AD (*cf* 3.1) on the program that discretizes and solves the direct system. This gives in fact the adjoint of the discrete function computed by the program. Theoretical results [22] guarantee convergence of these derivatives when the direct program converges. This approach is highly mechanizable, but leads to massive use of storage and may require code transformation by hand [27], [30] to reduce memory usage.

We study approaches between these extremes. If for instance the model is steady, one can use the iterated states in the direct order [24], or one can use only the fully converged final state. Since these mixed approaches can also be error-prone, we advocate incorporating them into the AD model and into the AD tools.

4. Application Domains

4.1. Panorama

Automatic Differentiation of programs gives sensitivities or gradients, that are useful for many types of applications:

- optimum shape design under constraints, multidisciplinary optimization, and more generally any algorithm based on local linearization,
- inverse problems, such as parameter estimation and in particular 4Dvar data assimilation in climate sciences (meteorology, oceanography),
- first-order linearization of complex systems, or higher-order simulations, yielding reduced models for simulation of complex systems around a given state,
- mesh adaptation and mesh optimization with gradients or adjoints,
- equation solving with the Newton method,
- sensitivity analysis, propagation of truncation errors.

4.2. Multidisciplinary optimization

A CFD program computes the flow around a shape, starting from a number of inputs that define the shape and other parameters. From this flow, it computes an optimization criterion, such as the lift of an aircraft. To optimize the criterion by a gradient descent, one needs the gradient of the output criterion with respect to all the inputs, and possibly additional gradients when there are constraints. The reverse mode of AD is a promising way to compute these gradients.

4.3. Inverse problems and Data Assimilation

Inverse problems aim at estimating the value of hidden parameters from other measurable values, that depend on the hidden parameters through a system of equations. For example, the hidden parameter might be the shape of the ocean floor, and the measurable values the altitude and speed of the surface.

One particular case of inverse problems is *data assimilation* [28] in weather forecasting or in oceanography. The quality of the initial state of the simulation conditions the quality of the prediction. But this initial state is largely unknown. Only some measures at arbitrary places and times are available. A good initial state is found by solving a least squares problem between the measures and a guessed initial state which itself must verify the equations of meteorology. This boils down to solving an adjoint problem, which can be done though AD [31]. Figure 1 shows an example of a data assimilation exercise using the oceanography code OPA [29] and its AD adjoint code produced by TAPENADE.

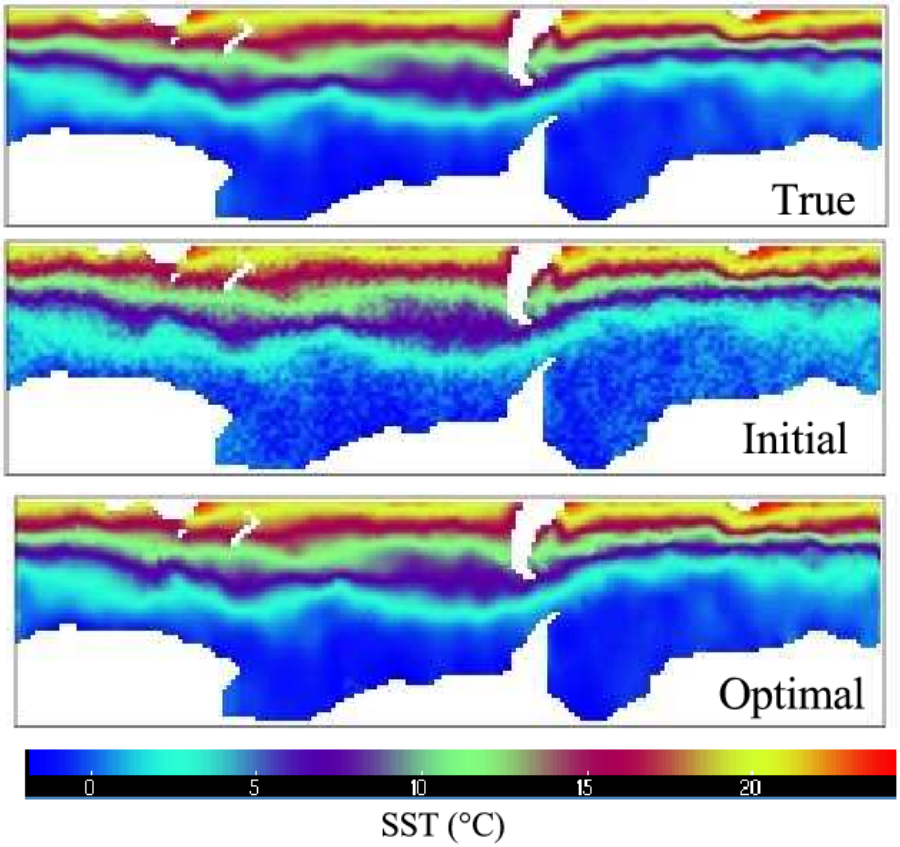

Figure 1. Twin experiment using the adjoint of OPA. We add random noise to a simulation of the ocean state around the Antarctic, and we remove this noise by minimizing the discrepancy with the physical model

The special case of *4Dvar* data assimilation is particularly challenging. The 4^{th} dimension in "4D" is time, as available measures are distributed over a given assimilation period. Therefore the least squares mechanism must be applied to a simulation over time that follows the time evolution model. This process gives a much better estimation of the initial state, because both position and time of measurements are taken into account. On the other hand, the adjoint problem involved grows in complexity, because it must run (backwards) over many time steps. This demanding application of AD justifies our efforts in reducing the runtime and memory costs of AD adjoint codes.

4.4. Linearization

Simulating a complex system often requires solving a system of Partial Differential Equations. This is sometimes too expensive, in particular in the context of real time. When one wants to simulate the reaction of this complex system to small perturbations around a fixed set of parameters, there is a very efficient approximate solution: just suppose that the system is linear in a small neighborhood of the current set of parameters. The reaction of the system is thus approximated by a simple product of the variation of the parameters with the Jacobian matrix of the system. This Jacobian matrix can be obtained by AD. This is especially cheap when the Jacobian matrix is sparse. The simulation can be improved further by introducing higher-order derivatives, such as Taylor expansions, which can also be computed through AD. The result is often called a *reduced model*.

4.5. Mesh adaptation

Some approximation errors can be expressed by an adjoint state. Mesh adaptation can benefit from this. The classical optimization step can give an optimization direction not only for the control parameters, but also for the approximation parameters, and in particular the mesh geometry. The ultimate goal is to obtain optimal control parameters up to a precision prescribed in advance.

5. Software

5.1. AIRONUM

Participant: Alain Dervieux [correspondant].

AIRONUM is an experimental software that solves the unsteady compressible Navier-Stokes equations with K-epsilon, LES-VMS and hybrid turbulence modelling on parallel platforms with Mpi as parallel programming concept. The mesh model is unstructured tetrahedrization, with possible mesh motion.

See also the web page http://www-sop.inria.fr/tropics/aironum.

- Version: v 1.0
- Programming language: FORTRAN95 (mostly). About 100,000 lines.

AIRONUM was developed by INRIA and university of Montpellier. It is used by INRIA, university of Montpellier and university of Pisa (I). AIRONUM is used as an experimental platform for:

- Numerical approximation of compressible flows, such as upwind mixed element volume approximation with superconvergence on regular meshes.
- Numerical solution algorithms for the implicit time advancing of the compressible Navier-Stokes equations, such as parallel scalable deflated additive Schwarz algorithms.
- Turbulence modelling such as the Variational Multiscale Large eddy Simulation and its hybridization with RANS statistical models.

5.2. TAPENADE

Participants: Laurent Hascoët [correspondant], Valérie Pascual.

TAPENADE is an Automatic Differentiation tool that transforms an original source program into a new source program that computes derivatives of the original program. Automatic Differentiation produces analytical derivatives, that are exact up to machine precision. The reverse mode of Automatic Differentiation is able to compute gradients at a cost which is independent from the number of input variables. TAPENADE accepts source programs written in Fortran 77, Fortran 90, or C.It provides differentiation in the following modes: tangent, multi-directional tangent, and reverse. Documentation is provided on the web site of the reserch team and as the INRIA technical report RT-0300. TAPENADE runs under Linux or Windows operating systems, and requires installation of Java jdk1.6 or upward.

See also the web page http://www-sop.inria.fr/tropics/.

- Version: v 3.6, september 2011
- ACM: D.3.4 Compilers; G.1.0 Numerical algorithms; G.1.4 Automatic differentiation; I.1.2 Analysis of algorithms
- AMS: 65K10; 68N20
- APP: IDDN.FR.001.040038.000.S.P.2002.000.31235
- Keywords: automatic differentiation, adjoint, gradient, optimisation, inverse problems, static analysis, data-flow analysis, compilation
- Programming language: Java

TAPENADE implements the results of our research about models and static analyses for AD. TAPENADE is can be downloaded and installed on most architectures. Alternatively, it can be used as a web server. TAPENADE differentiates computer programs according to the model described in section 3.1 Higher-order derivatives can be obtained through repeated application of tangent AD on tangent and/or reverse AD.

TAPENADE performs sophisticated data-flow analysis, flow-sensitive and context-sensitive, on the complete source program to produce an efficient differentiated code. Analyses include Type-Checking, Read-Write analysis, and Pointer analysis. AD-specific analysis include:

- **Activity analysis:** This detects variables whose derivative is either null or useless, to reduce the number of derivative instructions.
- **Adjoint Liveness analysis:** This detects the source statements that are dead code for the computation of derivatives.
- **TBR analysis:** In reverse mode, this reduces the set of source variables that need to be recovered.

TAPENADE is not open-source. Academic usage is free. Industrial or commercial usage require a paying license, as detailled on the team's web page. The software has been downloaded several hundred times, and the web tool served several thousands of true connections (not robots). The tapenade-users mailing list is over one hundred registered users.

6. New Results

6.1. Automatic Differentiation and parallel codes

Participants: Valérie Pascual, Laurent Hascoët, Hubert Alcin, Jean Utke [Argonne National Lab. (Illinois, USA)], Uwe Naumann [RWTH Aachen University (Germany)].

This research is an ongoing joint work between three teams working on AD. We study differentiation in reverse mode of programs that contain MPI communication calls. Instead of the commonly used approach that encapsulates the MPI calls into black-box subroutines that will be differentiated by hand, we are looking for a native differentiation of the MPI calls by the AD tool. The ultimate goal of this work is to generate the adjoint of MPI-parallel codes.

One issue is to reduce the variability of the available MPI procedures and parameters to a smaller number of elementary concepts. We then address the basic question of `sends` and `recvs`, that may be blocking or nonblocking, individual or collective, and so on. Essentially the adjoint of a `send` is a `recv`, and vice-versa, but the possibility of nonblocking `isend`'s and `irecv`'s requires more subtlety.

We continue adaptation of the tool's static analysis to programs with message-passing communication. In the framework of flow-sensitive and context-sensitive data-flow analysis, we introduce new "channel" variables and modify the general static data propagation mechanism.

Implementation in TAPENADE is well advanced: the main MPI procedures are now correctly understood and all data-flow analyses are adapted. In 2011, we obtained a first valid tangent differentiated code of the team's CFD code AIRONUM. Work is continuing to obtain the adjoint.

Our team focuses on AD based on program transformation. On the other hand, we closely follow the developments of operator-overloading AD tools towards message-passing communication. This requires a more complex definition of the overloaded communication primitives [33].

6.2. AD adjoints and Dynamic Memory

Participants: Laurent Hascoët, Jean Utke [Argonne National Lab. (Illinois, USA)].

Again in the adjoint mode, dynamic memory allocation and the associated pointer manipulations pose difficult problems. As the adjoint mode is bound to recover past values of variables, the need arises to recover past values of pointers too. However, these values are addresses often relative to some dynamically allocated memory. The original program often manages memory by allocating and deallocating memory on the run. Correspondingly, the adjoint program will have to reallocate memory as it walks backwards along the original allocation. If recovery of past values is implemented through storage, e.g. on a stack, the stored addresses refer to memory that has been deallocated, and do not correspond to the reallocated memory zones.

In 2011 we investigated this problem in two principal directions:

- We may consider not storing the address, but rather recompute it by repeating in the reverse sweep its calculation from the forward sweep. This is not always possible as pointer assignment and manipulation may be complex and distributed in the code. However, when possible, it is certainly a very efficient approach. We propose an data-flow algorithm to detect applicable cases.
- For the remaining cases where storage must be used, we study an address mapping mechanism, such that addresses inside some allocated memory can be dynamically converted into addresses inside the corresponding reallocated memory.

6.3. Resolution of linearised systems

Participants: Hubert Alcin, Olivier Allain [Lemma], Anca Belme, Marianna Braza [IMF-Toulouse], Alexandre Carabias, Alain Dervieux, Bruno Koobus [Université Montpellier 2], Carine Moussaed [Université Montpellier 2], Hilde Ouvrard [IMF-Toulouse], Stephen Wornom [Lemma].

The interaction between the sophisticated solution algorithm inside a program and the Automatic Differentiation of the program is a non-trivial issue. An iterative algorithm generally does not store the successive updates of the iterated solution vector. Furthermore, a modern iterative solution algorithm involves several nonlinear processes, like in:

- the evaluation of an optimal step, which results at least from a homographic function of the unknown,
- the orthonormalisation of the updates (Gram-Schmidt method, Hessenberg method).

Applying reverse AD to the iterative solution algorithm produces a *linearised iterative algorithm* which is transposed and therefore follows a reverse order, with exactly the same number of iterations, and needing exactly each of the iterated state solution vectors. This effect is considerably amplified in the case of the numerical simulation of unsteady phenomena with implicit numerical schemes. For example, the simulation of high Reynolds turbulent flows by a Large Eddy Simulation (LES) requires hundreds of thousands time steps, each of them involving a modern iterative solution algorithm.

In the 4-year ECINADS ANR project, we design more efficient solution algorithms and we examine the questions risen by their reverse differentiation. The application domain is the computation of high Reynolds turbulent flows with LES and hybrid RANS-LES models. The efficiency will be evaluated through the practical scalability on a large number of processors. This efficiency criterion also extends to the scalability of the reverse/adjoint algorithm. ECINADS also addresses the scalable solution of new approximations. ECINADS associates the university of Montpellier 2, the Institut de Mécanique des fluides de Toulouse and Lemma company. The kick off meeting of ECINADS was held at end of 2009.

In 2011, Hubert Alcin has performed a study of deflation and balancing coarse grid methods for a set of scalar models. The methods has been extended to the incompressible Navier-Stokes model in Lemma's software ANANAS by Olivier Allain and to compressible Navier-Stokes by Bruno Koobus and Hubert Alcin. A collection of benchmark tests on these models has been performed and show a good scalability for the tested algorithms. An article is prepared on these results. Hubert Alcin has also studied a novel method for three-level preconditioning. The new method will be extended to compressible flows in cooperation with the Montpellier team (Carine Moussaed and Bruno Koobus).

6.4. Perturbation Methods

Participants: Alain Dervieux, Laurent Hascoët.

In the context of the European project NODESIM-CFD, the contribution of Tropics involved mainly the differentiation of perturbation methods and reduced order models for the management of uncertainties. These methods rely on Taylor series with second-order terms. The production of second derivative code is obtained through repeated application of Automatic Differentiation. Three strategies can be applied to obtain (elements of) the Hessian matrix, named Tangent-on-Tangent (ToT), Tangent-on-Reverse (ToR), and Reverse-on-Tangent (RoT). These new methods are disseminated through short courses, as those given by Alain Dervieux at ERCOFTAC sessions (Munich and Hampton). The application and extension of these methods have motivated a joint application from the Italian Aircraft company Alenia and Tropics in a FP7 proposal (Proposal CARDINA, nov. 2011).

6.5. Control of approximation errors

Participants: Frédéric Alauzet [GAMMA team, INRIA-Rocquencourt], Estelle Mbinky [GAMMA team, INRIA-Rocquencourt], Olivier Allain [Lemma], Anca Belme, Alexandre Carabias, Hubert Alcin, Alain Dervieux.

This is a joint research between three INRIA teams GAMMA (Rocquencourt), TROPICS, PUMAS and Lemma company. Roughly speaking, GAMMA brings mesh and approximation expertise, TROPICS contributes to adjoint methods, and CFD applications are developed in the context of PUMAS and Lemma.

The resolution of the optimum problem using the innovative approach of an AD-generated adjoint can be used in a slightly different context than the optimal shape design, namely the mesh adaptation. This will be possible if we can map the mesh adaptation problem into a differentiable optimal control problem. To this end, we have introduced a new methodology that consists in stating the mesh adaptation problem in a purely functional form: the mesh is reduced to a continuous property of the computational domain: the continuous metric. We minimize a continuous model of the error resulting from that metric. Thus the problem of searching an adapted mesh is transformed into the search of an optimal metric.

In 2011, a work on goal-oriented mesh adaptation for unsteady Euler flows has been extended, with further analysis, and a paper has been written and submitted to a journal. Its extension to the compressible Navier-Stokes model has been developed, [11] and a paper is being written. A further extension to Large Eddy Simulation is started. The method is being extended to a third-order approximation, the Vertex-CENO. This approximation was defined during a collaboration between university of Montpellier, IMM-Moscow and Tropics. A more accurate version has been studied by Alexandre Carabias and presented in Honom-Trento. a new theory involving error estimates and criteria has been developed by Gamma and Tropics. The extension of the multiscale adaptation method is considered by Estelle Mbinky at Rocquencourt. The extension of the goal-oriented method is considered by Alexandre Carabias at Sophia. Anisotropic mesh adaptation allows for better convergence to continuous solutions, and in particular more accurate a posteriori error estimates and correctors. The synergy between correctors and mesh adaptation is currently analysed and is the subject of a joint contribution (Gamma and Tropics) for the FP7 CARDINA proposal (nov. 2011).

7. Partnerships and Cooperations

7.1. International Initiatives

7.1.1. Visits of International Scientists

Trond Steihaug, professor at the University of Bergen, is spending a sabbatical period inside Tropics, from september 2011 to may 2012. He was driven to AD by his works on Truncated Newton methods.

8. Dissemination

8.1. Animation of the scientific community

- Anca Belme presented a talk on "A priori anisotropic goal-oriented estimates for mesh adaptation in compressible CFD" at the 16th International Conference on Finite Elements in Flow Problems (march).
- Alain Dervieux gave a short course on "Sensitivity analysis by adjoint Automatic Differentiation and Application" at the ERCOFTAC Course on Uncertainty Management and Quantification in Munich, Germany (march), and again at the ERCOFTAC-NIA Course in Hampton, Virginia, USA (september).
- Alexandre Carabias and Alain Dervieux presented their work on "Dissipation and dispersion control of a quadratic-reconstruction advection scheme" at Honom 2011 in Trento, Italy (april).
- Alain Dervieux was in the PhD jury of Géraldine Olivier, Paris VI (april).
- Alexandre Carabias has presented a poster on "Analyse et amélioration d'un schéma à reconstruction quadratique" at SMAI 2011, Lorient (may).
- Anca Belme presented her work at University of Montpellier and at University of Pau.
- Alain Dervieux presented a communication on "The effect of consistent coarse grid in Schwarz algorithms" at Parallel-CFD-2011, Barcelona, Spain (may).
- Hubert Alcin presented a talk on "Introduction des grilles grossières dans la méthode de déflation et de balancing" in Montpellier at an ECINADS seminar, joint with the Barcelona Computing Center (may).
- Anca Belme presented a talk on "Application of anisotropic goal-oriented unsteady mesh adaptation to Aerodynamics and Aeroacoustics" at the ADMOS 2011 conference, Paris (june).
- Alain Dervieux gave a short course on "Indicateurs de raffinement et adaptation de maillage en simulation numérique pour la Mécanique des Fluides" at Collège X, Paris (june).
- Hubert Alcin presented a communication on "Volume-Agglomeration Coarse Grid In Schwarz Algorithm" to FVCA6, Prague, Czech (june).
- The team has an agreement with EADS in Suresnes, France, to support their use of AD in inverse problems. Laurent Hascoët visited them twice in 2011.
- Trond Steihaug presented his work on "A Class of Methods Combining L-BFGS and Truncated Newton" at the 26th International Symposium on Computer and Information Sciences, Royal Society, London, UK (september).
- Frederic Alauzet presented joint work on "Anisotropic goal-oriented mesh adaptation for time dependent problems" at the 20th IMR Conference in Paris (october).
- Laurent Hascoët was in the PhD jury of P. Pham Quang, INPG Grenoble (october).

- Laurent Hascoët gave a short course on "Automatic Differentiation in Optimization" at the ERCOFTAC course on Design Optimization in Manching, Germany (november).
- Alain Dervieux was referee in the PhD jury of Thibaud Marcel, University of Toulouse (november).
- Alain Dervieux attended a seminar of the Groupement de Recherche Interaction Fluid Structure, in Sophia-Antipolis (november).
- Laurent Hascoët is on the organizing commitee of the Euro AD Workshops. He attented this year's workshop in Berlin, Germany (december).
- Trond Steihaug gave a talk on "Factorable Programming revisited" at the 12th Euro AD Workshop in Berlin, Germany (december).
- Laurent Hascoët is a member of the internal "CDT" committee at INRIA Sophia-Antipolis ("Comité Développement Technologique").
- TROPICS is coordinator of the ANR project ECINADS, with PUMAS team, university Montpellier 2, Institut de mécanique des Fluides de Toulouse and the Lemma company in Sophia-Antipolis. ECINADS concentrates on solution algorithms for state and adjoint systems in CFD.

8.2. Teaching

Anca Belme: "Algorithmique Numérique", 52 h, 1st year école d'ingénieur, Université de Nice.

Hubert Alcin: "Algorithmique Numérique", 45 h, 1st year école d'ingénieur, Université de Nice.

Hubert Alcin: "Outils mathématiques pour l'ingénieur", 20 h, 1st year prépa intégrée, Université de Nice.

Laurent Hascoët: "Optimisation Avancée", 10 h, Master 2, Université de Nice.

PhD & HdR

PhD : Anca Belme, "Aérodynamique instationnaire et méthode de l'adjoint", Université de Nice, defended december 8, 2011, advisor Alain Dervieux.

PhD in progress : Hubert Alcin, "Contribution algorithmique aux schémas hybrides pour la simulation des grandes structures d'écoulements turbulents", started october 2009, advisor Alain Dervieux.

PhD in progress : Alexandre Carabias, "Algorithmes parallèles et adaptation de maillages pour des phénomènes thermiques complexes", started october 2010, advisor Alain Dervieux.

9. Bibliography

Major publications by the team in recent years

[1] F. COURTY, A. DERVIEUX. *Multilevel functional Preconditioning for shape optimisation*, in "International Journal of CFD", 2006, vol. 20, n^{O} 7, p. 481-490.

[2] F. COURTY, A. DERVIEUX, B. KOOBUS, L. HASCOËT. *Reverse automatic differentiation for optimum design: from adjoint state assembly to gradient computation*, in "Optimization Methods and Software", 2003, vol. 18, n^{O} 5, p. 615-627.

[3] B. DAUVERGNE, L. HASCOËT. *The Data-Flow Equations of Checkpointing in reverse Automatic Differentiation*, in "International Conference on Computational Science, ICCS 2006, Reading, UK", 2006.

[4] A. GRIEWANK. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM, Frontiers in Applied Mathematics, 2000.

[5] L. HASCOËT, M. ARAYA-POLO. *The Adjoint Data-Flow Analyses: Formalization, Properties, and Applications*, in "Automatic Differentiation: Applications, Theory, and Tools", H. M. BÜCKER, G. CORLISS, P. HOVLAND, U. NAUMANN, B. NORRIS (editors), Lecture Notes in Computational Science and Engineering, Springer, 2005.

[6] L. HASCOËT, S. FIDANOVA, C. HELD. *Adjoining Independent Computations*, in "Automatic Differentiation of Algorithms: From Simulation to Optimization", New York, NY, G. CORLISS, C. FAURE, A. GRIEWANK, L. HASCOËT, U. NAUMANN (editors), Computer and Information Science, Springer, New York, NY, 2001, chap. 35, p. 299-304.

[7] L. HASCOËT, U. NAUMANN, V. PASCUAL. *"To Be Recorded" Analysis in Reverse-Mode Automatic Differentiation*, in "Future Generation Computer Systems", 2004, vol. 21, n° 8.

[8] L. HASCOËT, J. UTKE, U. NAUMANN. *Cheaper Adjoints by Reversing Address Computations*, in "Scientific Programming", 2008, vol. 16, n° 1, p. 81–92.

[9] L. HASCOËT, M. VÁZQUEZ, B. KOOBUS, A. DERVIEUX. *A Framework for Adjoint-based Shape Design and Error Control*, in "Computational Fluid Dynamics Journal", 2008, vol. 16, n° 4, p. 454-464.

[10] M. VÁZQUEZ, A. DERVIEUX, B. KOOBUS. *Multilevel optimization of a supersonic aircraft*, in "Finite Elements in Analysis and Design", 2004, vol. 40, p. 2101-2124.

Publications of the year

Doctoral Dissertations and Habilitation Theses

[11] A. BELME. *Unsteady Aerodynamics and Adjoint method*, Université de Nice Sophia-Antipolis, 2011.

Articles in International Peer-Reviewed Journal

[12] R. BOURGUET, M. BRAZA, A. DERVIEUX. *Reduced order modeling of transonic flows around an airfoil*, in "Journal of Computational Physics", 2011, vol. 230, n° 1, p. 159-184.

[13] M.-V. SALVETTI, H. OUVRARD, B. KOOBUS, S. WORNOM, S. CAMARRI, A. DERVIEUX. *Simulation of the flow past a circular cylinder in the supercritical regime by blending RANS and variational-multiscale LES approaches*, in "Flow, Turbulence and Combustion", 2011, to appear.

[14] S. WORNOM, H. OUVRARD, M.-V. SALVETTI, B. KOOBUS, A. DERVIEUX. *Classical and Variational Multiscale Large-Eddy Simulations of the Flow past a Circular Cylinder : Reynolds number effects*, in "Computer and Fluids", 2011, vol. 47, p. 44-50.

Research Reports

[15] S. WORNOM. *Analysis of flow over cylinders using the AIRONUM software*, INRIA, March 2011, n° RR-7550, http://hal.inria.fr/inria-00574198/en.

References in notes

[16] A. AHO, R. SETHI, J. ULLMAN. *Compilers: Principles, Techniques and Tools*, Addison-Wesley, 1986.

[17] I. ATTALI, V. PASCUAL, C. ROUDET. *A language and an integrated environment for program transformations*, INRIA, 1997, n° 3313, http://hal.inria.fr/inria-00073376.

[18] D. CLÉMENT, J. DESPEYROUX, L. HASCOËT, G. KAHN. *Natural semantics on the computer*, in "K. Fuchi and M. Nivat, editors, Proceedings, France-Japan AI and CS Symposium, ICOT", 1986, p. 49-89, Also, Information Processing Society of Japan, Technical Memorandum PL-86-6. Also INRIA research report # 416, http://hal.inria.fr/inria-00076140.

[19] J.-F. COLLARD. *Reasoning about program transformations*, Springer, 2002.

[20] P. COUSOT. *Abstract Interpretation*, in "ACM Computing Surveys", 1996, vol. 28, n° 1, p. 324-328.

[21] B. CREUSILLET, F. IRIGOIN. *Interprocedural Array Region Analyses*, in "International Journal of Parallel Programming", 1996, vol. 24, n° 6, p. 513–546.

[22] J. GILBERT. *Automatic differentiation and iterative processes*, in "Optimization Methods and Software", 1992, vol. 1, p. 13–21.

[23] M.-B. GILES. *Adjoint methods for aeronautical design*, in "Proceedings of the ECCOMAS CFD Conference", 2001.

[24] A. GRIEWANK, C. FAURE. *Reduced Gradients and Hessians from Fixed Point Iteration for State Equations*, in "Numerical Algorithms", 2002, vol. 30(2), p. 113–139.

[25] A. GRIEWANK, A. WALTHER. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, 2nd, SIAM, Other Titles in Applied Mathematics, 2008.

[26] L. HASCOËT. *Transformations automatiques de spécifications sémantiques: application: Un vérificateur de types incremental*, Université de Nice Sophia-Antipolis, 1987.

[27] P. HOVLAND, B. MOHAMMADI, C. BISCHOF. *Automatic Differentiation of Navier-Stokes computations*, Argonne National Laboratory, 1997, n° MCS-P687-0997.

[28] F.-X. LEDIMET, O. TALAGRAND. *Variational algorithms for analysis and assimilation of meteorological observations: theoretical aspects*, in "Tellus", 1986, vol. 38A, p. 97-110.

[29] G. MADEC, P. DELECLUSE, M. IMBARD, C. LEVY. *OPA8.1 ocean general circulation model reference manual*, Pole de Modelisation, IPSL, 1998.

[30] B. MOHAMMADI. *Practical application to fluid flows of automatic differentiation for design problems*, in "Von Karman Lecture Series", 1997.

[31] N. ROSTAING. *Différentiation Automatique: application à un problème d'optimisation en météorologie*, université de Nice Sophia-Antipolis, 1993.

[32] R. RUGINA, M. RINARD. *Symbolic Bounds Analysis of Pointers, Array Indices, and Accessed Memory Regions*, in "Proceedings of the ACM SIGPLAN'00 Conference on Programming Language Design and Implementation", ACM, 2000.

[33] M. SCHANEN, U. NAUMANN, L. HASCOËT, J. UTKE. *Interpretative Adjoints for Numerical Simulation Codes using MPI*, in "Proceedings of the 10th International Conference on Computational Science, ICCS'2010", 2010.