COMPUTATION OF EXACT GRADIENTS IN DISTRIBUTED DYNAMIC SYSTEMS[1]

Yuri EVTUSHENKO

Computing Center of the Russian Academy of Sciences
40, Vavilov Street, 117967 Moscow, Russia

(Received 28 August 1995; in final form 2 June 1997)

Abstract. A new and unified methodology for computing first order derivatives of functions obtained in complex multistep processes is developed on the basis of general expressions for differentiating a composite function. From these results, we derive the formulas for fast automatic differentiation of elementary functions, for gradients arising in optimal control problems, nonlinear programming and gradients arising in discretizations of processes governed by partial differential equations. In the proposed approach we start with a chosen discretization scheme for the state equation and derive the exact gradient expression. Thus a unique discretization scheme is automatically generated for the adjoint equation. For optimal control problems, the proposed computational formulas correspond to the integration of the adjoint system of equations that appears in Pontryagin's maximum principle. This technique appears to be very efficient, universal, and applicable to a wide variety of distributed controlled dynamic systems and to sensitivity analysis.

Keywords: Fast automatic differentiation; optimal control problem; differentiation of elementary functions; rounding error estimation; parabolic system; hyperbolic system; adjoint equation; sensitivity analysis.

1 INTRODUCTION

In 1970-1980 we developed a number of numerical methods and software for solving practical optimal control problems. Our technique was based on the reduction of the original problem to the nonlinear programming problem (NLP). This approach turned out to be very efficient for many reasons. It allowed the use of numerous sophisticated NLP methods; many earlier heuristic methods developed for optimal control became evident; it permitted us to solve complicated problems with delay, and with mixed phase and control constraints. We obtained formulas for computing the first and second derivatives of the cost functional for the integration by Euler and Runge-Kutta formulas of a system of ordinary differential equations that describe a controlled process. Using these results, we created the library of algorithms for solving a wide class of optimal control problems [7, 8].

This paper extends our technique further to general complex computational processes. We apply the approach to the optimization of distributed dynamic systems. The mathematical model of a controlled system and its initial and boundary conditions are approximated by a corresponding optimal parameter selection problem which, in turn, is viewed as a mathematical programming problem and hence can be solved by existing optimization algorithms. Using our formulas, we derive the expression for the exact gradient of the objective functional. Once this is accomplished, efficient gradient based algorithms for solving mathematical programming problems can be easily applied to solving the parameter selection problem. These gradient algorithms usually require significantly fewer iterations and function evaluations than derivative-free methods which only use function values.

Our computational formulas were similar to ones developed for fast automatic differentiation (FAD), used for differentiating multivariate functions. Many publications have been devoted to the FAD technique. We refer the reader to the proceedings of the first SIAM Workshop on the Automatic Differentiation of Algorithms held in Breckenridge, Colorado in 1991 (see [10]). An overview of the history and the state of the art in automatic differentiation and related techniques is given by Iri in [12]. In many cases, FAD is far superior to symbolic differentiation or to divided differences approximation.

Having studied literature on FAD, we realized that the expressions which we used for computing a gradient included the FAD formulas as a special case, where an explicitly defined elementary function was differentiated. It turned out that our formulas are more general and could be used for gradient calculations in both explicitly and implicitly given functions and for computational processes that arise from discrete approximations of continuous

[1]Research was supported by grants N 95-01-00779 and N 96-15-96124 from the Russian Foundation for Basic Research, by FAPESP grant N 1996/6631-5, sponsored by Russian Institute for Mechanics of Smart Materials. Some of this research was carried out during the author's visit to the Mathematics Department at the University of Western Australia, which was partially supported by a research grant of K.L. Teo from the Australian Research Council.

systems governed by ordinary and partial differential equations. Later we called these results "generalized FAD formulas". Our preliminary investigations in this field were published in [1, 5, 6, 7, 8].

In Section 2, we present general expressions from which we obtain the so-called "forward" and "reverse" differentiation expressions as a special case. We introduce a new auxiliary function and use a canonical form which turns out to be very convenient for the representing the generalized reverse differentiation formulas.

In Section 3, we consider the algebraic complexity of computing a function of several variables and its partial derivatives with respect to all the variables. If FAD is used, then the ratio of the computer time for calculating all partial derivatives of an elementary function to the time for calculating the underlying function value is bounded above by 3.

In Section 4, we derive expressions for the gradient of a function defined by the solution of an initial value problem for ordinary differential equation. We discretize the problem and, applying the expressions from Section 2, we obtain the gradient by using forward and reverse computations. Going from the discrete to continuous case, we obtain the gradient expressions well known in the theory of ordinary differential equations.

Similar results for optimal control problems are given in Section 5, where we describe a general approach to approximating the infinite-dimensional optimization problem by a finite-dimensional problem. The dynamic equations are discretized and the optimal control problem is transformed into a nonlinear programming problem, which is solved by direct gradient numerical methods. We show that the reverse computation of the gradient corresponds to integrating the adjoint system of equations that appears in Pontryagin's maximum principle.

Our approach permits us to develop a new methodology for calculating the gradient in a controlled system described by a partial differential equations (PDE) which we call the "state equation". In Sections 6 and 7, we derive the required exact functional gradient using the general expressions given in Section 2. We show how to apply this technique in two simple cases of parabolic and hyperbolic controlled equations. The adjoint variable formulations for computing the gradients for continuous systems were obtained in [21, 22, 23] based on calculus of variations. In these and many similar papers, the adjoint equation technique is employed. The adjoint equations are derived in the PDE form. This classic approach has a drawback — it is not clear how the state and adjoint equations should be discretized in order to find the exact functional gradient for a chosen discrete approximation of the state equation.

The difficulties in integrating the state PDE and adjoint PDE independently are described in numerous papers. For example, in [16] the authors write, "it is difficult to pass from the continuous to the discrete formulation, especially for nonlinear advection terms. This has led many researchers to use methods working exclusively from the discrete equations of the model... it can be noticed that the problems to find the "good" gradient arise from some of the nonlinear terms of the equation." In [16], a special "matrix approach" was proposed to overcome these difficulties.

In our approach, we start with a chosen discretization scheme for the state PDE and initial and boundary conditions, then we derive the *exact* gradient expressions which involve adjoint variables. Thus we automatically generate a unique discretization scheme for the PDE formulation of the adjoint equation. This technique appears to be very efficient, universal, and it could be used for differentiation in numerous complicated distributed systems and for sensitivity analysis.

2 GENERAL EXPRESSIONS

There are many ways to derive the formulas for differentiation of a composite function. Among these the shortest and most general way is based on the well-known implicit function theorem. Suppose the mappings $\Phi : \mathbb{R}^n \times \mathbb{R}^r \to \mathbb{R}^n$ and $W : \mathbb{R}^n \times \mathbb{R}^r \to \mathbb{R}^1$ are differentiable. Let $z \in \mathbb{R}^n$ and $u \in \mathbb{R}^r$ satisfy the following nonlinear system of n scalar algebraic equations:

$$\Phi(z, u) = 0_n, \tag{1}$$

where 0_s is the s-dimensional null vector.

We will use the following notation. For $f : \mathbb{R}^n \to \mathbb{R}^m$, let f_x^T denote the $n \times m$-matrix, which ij-th element is equal to $\partial f^j / \partial x^i$, and mean the first derivative of the vector-row f^T over the vector-column x. When $m = 1$, the transposition sign in the vector-column f_x is omitted. We use partial derivative for differentiation of an explicitly given function (e.g. $\partial f / \partial x = f_x$) and total derivative for differentiation of implicitly definite composite function (e.g., $d\Omega/du$).

We assume that the matrix $\Phi_z^T(z, u)$ is nonsingular. According to the implicit function theorem, system (1) defines a continuous vector-valued function $z = z(u)$ which is differentiable and whose derivative dz^T/du, denoted by $N(u) \in \mathbb{R}^{r \times n}$, satisfies the following linear algebraic system:

$$\Phi_u^T(z(u), u) + N(u)\Phi_z^T(z(u), u) = 0_{rn}, \tag{2}$$

where $0_{\alpha\beta}$ is the $\alpha \times \beta$ null matrix. Thus,

$$N(u) = -\Phi_u^T(z(u), u)[\Phi_z^T(z(u), u)]^{-1}. \quad (3)$$

As a rule, z and u are referred to as *dependent* and *independent* variables, respectively. The composite function $\Omega(u) = W(z(u), u)$ is differentiable and the gradient with respect to the independent variable u (the reduced gradient) is equal to

$$d\Omega(u)/du = W_u(z(u), u) + N(u)W_z(z(u), u). \quad (4)$$

We introduce the Lagrange function $L(z, u, p) = W(z, u) + \Phi^T(z, u)p$ with the Lagrange multiplier $p \in \mathbb{R}^n$, which is required to satisfy the linear algebraic system:

$$L_z(z(u), u, p) = W_z(z(u), u) + \Phi_z^T(z(u), u)p = 0_n. \quad (5)$$

If the matrix $\Phi_z^T(z(u), u)$ is nonsingular, then the vector p is uniquely defined by (5) and the formula (4) can be rewritten in the form

$$d\Omega(u)/du = W_u(z(u), u) + \Phi_u^T(z(u), u)p = L_u(z(u), u, p). \quad (6)$$

Expressions (4) and (6) are mathematically equivalent, but from the computational point of view there is a crucial difference. A slight variation in the way the function is differentiated will result in a drastic change in the efficiency of computation. In the first case we use the auxiliary matrix N; in the second case we use an additional Lagrange vector p. We shall show that expression (4) corresponds to the so-called forward differentiation, and formula (6) — to reverse differentiation.

Instead of system (1), let us consider a perturbed system

$$\Phi(z, u) + \varepsilon = 0_n,$$

where ε is an n-dimensional vector.

Assume that this system defines a continuous function $z = z(u, \varepsilon)$. Taking into account (5), we obtain that the Lagrange multiplier $p(\varepsilon)$ satisfies the following condition

$$W_z(z(u, \varepsilon), u) + \Phi_z^T(z(u, \varepsilon), u)p(\varepsilon) = 0_n.$$

In view of (5), we have $p(0_n) = p$. Then, using (6), we find

$$d\Omega(u, \varepsilon)/d\varepsilon = p(\varepsilon).$$

That is, the vector p that satisfies (5) is the sensitivity of the composite function $\Omega(u, \varepsilon) = W(z(u, \varepsilon), u)$ to changes in system (1), if the derivative is evaluated at the point $\varepsilon = 0_n$.

Similar result was obtained in nonlinear programming (e.g., [3, p.70]) with an additional condition $d\Omega(u, \varepsilon)/du = 0_r$. Our results do not require this condition. Indeed, let us consider the following nonlinear programming problem:

$$\text{minimize } f(x) \text{ subject to } g(x) + \varepsilon = 0_n,$$

where $x \in \mathbb{R}^c$, $c = n + r$, $\varepsilon \in \mathbb{R}^n$, $f : \mathbb{R}^c \to \mathbb{R}$, $g : \mathbb{R}^c \to \mathbb{R}^n$.

Here $x^T = [z^T, u^T]$ and $f(x) = W(z, u)$, $g(x) = \Phi(z, u)$. A well-known constraint qualification (CQ) in NLP is regularity, which requires linear independence of the columns of the constraint gradient matrix $g_x^T(x)$ at feasible points. We use a weaker CQ, namely that the matrix $g_z^T(z(u, \varepsilon), u)$ be nonsingular. If this CQ holds, then the Lagrange multiplier $p(\varepsilon)$ is uniquely defined. Substituting $z(u, \varepsilon)$ and $p(\varepsilon)$ for $z(u)$ and p, respectively, we find from (6) the total derivative of $W(z(u, \varepsilon), u)$ with respect to u. The derivative is zero only in stationary points $u = u_*$. As a consequence of the expressions above, the derivative of $W(z(u, \varepsilon), u)$ with respect to ε is equal to $p(\varepsilon)$ everywhere in a neighborhood of a stationary point u_*. This is another distinction from the conventional NLP analysis.

As we will show, all the formulas derived above can be viewed as a basis for a number of expressions for calculating exact gradients in various complex problems and can be used in numerous minimization algorithms.

In multistep problems, the variables z and u are usually naturally partitioned into k variables of lower dimensionality:

$$z^T = [z_1^T, z_2^T, \ldots, z_k^T], \quad u^T = [u_1^T, u_2^T, \ldots, u_k^T], \quad z_i \in \mathbb{R}^s, \quad u_i \in \mathbb{R}^m, \quad 1 \le i \le k.$$

Under this assumption, relation (1) is split into k relations as follows:

$$z_i = F(i, Z_i, U_i), \quad 1 \le i \le k, \quad n = s \cdot k, \quad r = m \cdot k, \quad (7)$$

where Z_i and U_i are given sets of variables z_j and u_j, respectively, and the index i takes integer values from 1 to k. A more general case, when $i \in D \subset \{1, \ldots, k\}$, can be considered, but just for brevity, we suppose further that $D = \{1, \ldots, k\}$. For each $i \in D$, we introduce two sets of indices Q_i and K_i, containing the indices of all variables z_i and u_i belonging to the sets Z_i and U_i, respectively. Then

$$Q_i = \{j \in D : z_j \in Z_i\}, \quad K_i = \{j \in D : u_j \in U_i\}.$$

Let us introduce the *conjugate* index sets

$$\bar{Q}_i = \{j \in D : z_i \in Z_j\}, \quad \bar{K}_i = \{j \in D : u_i \in U_j\}$$

and the corresponding vector sets

$$\bar{Z}_i = \{z_j : j \in \bar{Q}_i\}, \quad \bar{U}_i = \{u_j : j \in \bar{K}_i\}.$$

The definition of these sets implies that if $z_j \in \bar{Z}_i,\ u_e \in \bar{U}_q$ (that is, if $j \in \bar{Q}_i, e \in \bar{K}_q$), then the following explicit functional dependencies are valid:

$$z_j = F(j, \ldots, z_i, \ldots), \quad z_e = F(e, \ldots, u_q, \ldots).$$

Therefore, the sets Q_i and K_i may be called the *input index sets at i-th step*, while $\bar{Q}_i$ and $\bar{K}_i$ are the *output index sets at i-th step*.

Consider in (1) the mapping $\Phi(z, u)$ composed of the mappings $F(j, Z_j, U_j) - z_j$, where $j \in D$. Denote $N_{ij} = dz_j^T/du_i \in \mathbb{R}^{m \times s}$. Then for process (7), we can rewrite (2) and (4) as follows:

$$N_{ij} = F_{u_i}^T(j, Z_j, U_j) + \sum_{q \in Q_j} N_{iq} F_{z_q}^T(j, Z_j, U_j), \tag{8}$$

$$d\Omega(u)/du_i = W_{u_i}(z, u) + \sum_{j \in D} N_{ij} W_{z_j}(z, u). \tag{9}$$

With multiplier vectors $p_j \in \mathbb{R}^s$ we introduce the new auxiliary function

$$E(z, u, p) = W(z, u) + \sum_{j \in D} F^T(j, Z_j, U_j) p_j.$$

Now, to consider the multistep process, let us turn from nonlinear programming to optimal control. In the last field the vectors z_i, u_i, and p_i are called state, control, and adjoint variables, respectively. In the sequel we shall use both terminologies.

It is very convenient to rewrite (7), (5), and (6) in the following *canonical* form:

$$z_i = E_{p_i}(z, u, p), \tag{10}$$

$$p_i = E_{z_i}(z, u, p) = W_{z_i}(z, u) + \sum_{q \in \bar{Q}_i} F_{z_i}^T(q, Z_q, U_q) p_q, \tag{11}$$

$$d\Omega(u)/du_i = E_{u_i}(z, u, p) = W_{u_i}(z, u) + \sum_{q \in \bar{K}_i} F_{u_i}^T(q, Z_q, U_q) p_q, \tag{12}$$

where $i \in D$.

We say that z_i is *an output vector in process* (7) if the index set $\bar{Q}_i$ is empty. In this case, $p_i = W_{z_i}(z, u)$.

If we regard each step of computational process (10) as a constraint with corresponding Lagrange multiplier p_i, then instead of E we can use the Lagrange function

$$L(z, u, p) = W(z, u) + \sum_{j \in D} [F^T(j, Z_j, U_j) - z_j^T] p_j.$$

The main expressions (10)—(12) can be rewritten as follows:

$$L_{p_i}(z, u, p) = 0_s, \quad L_{z_i}(z, u, p) = 0_s, \quad d\Omega/du_i = L_{u_i}(z, u, p).$$

The latter representation is traditionally employed for first-order necessary optimality conditions where the statement $L_{u_i}(z, u, p) = 0$ is added and all conditions acquire a symmetric form. Since the representation (10)—

(12) is based essentially on a particular structure of the mapping Φ given by (7), this representation suits our purposes better. Therefore, it will be used in sequel.

The multistep process (7) is said to be *explicit* if for every step $j \in D$ the input set Q_j is such that for any element $q \in Q_j$ the inequality $q < j$ holds. By definition, in this case two sets Q_1 and $\bar{Q}_k$ must be empty. According to (8) and (10), each matrix N_{ij} and each vector z_j can be expressed for explicit processes by means of the already computed in previous computational steps matrices N_{iq} and vectors z_q, respectively, where $1 \leq q < j$. Expression (9) yields all required derivatives. We say that all matrices N_{ij} and derivatives (9) are computed in *forward, or bottom-up, or from left to right, or contravariant mode.* During their computation the index j increases from 1 to k.

Let us show that if the multistep process (7) is explicit, then for all $i \in D$ the output set $\bar{Q}_i$ is such that for any element $q \in \bar{Q}_i$ the inequality $i < q$ holds. Assume the converse. Then there exists $q \in \bar{Q}_i$ such that $q \leq i$. It means that $i \in Q_q$. But taking into account that the process (7) is explicit we obtain that $i < q$. The contradiction proves the statement.

In the last computational step of explicit process (7) we obtain a vector z_k which does not belong to all sets Q_i, $1 \leq i \leq k$ and, therefore, z_k is output vector in process (7). From (11) we obtain

$$p_k = W_{z_k}(z, u). \tag{13}$$

Hence starting from $i = k$ we can sequentially find from (13), (11) all vectors p_i and using (12) compute all derivatives. We say that all vectors p_i and derivatives are found in the *reverse, or backward, or top-down, or from right to left, or covariant mode*, which means that i decreases from $i = k$ to $i = 1$. This makes the explicit processes very simple for computations, they are very often used for solving discrete optimal control problems, where continuous differential equations are integrated using explicit numerical schemes. But sometimes we have to deal with implicit formulas. The simplest example of such case is the process (34) given in Section 5.

There is an interesting graph representation for the computational process of evaluating functions and their derivatives (see for example [12]). The process is visualized figuratively and graphically. However, applying this technique to an implicit scheme is more complicated. There is a cyclic computational graph and we have to use our unified approach based on expressions (10)—(12), where we did not postulate the explicitness of the computational process (7). If implicit integration expressions are used, then at each step i we have to solve the system of nonlinear equations (7) and find the vector z_i. Next, from the linear algebraic systems (8) and (11), we obtain N_{ij} and p_i, respectively.

We should emphasize that there is an important difference between our expressions (8)—(12) and the well–known "forward" and "reverse" differentiation formulas. Our expressions reduce to the latter when process (7) is explicit. In the general case, our expressions could not be called "forward" or "reverse". The difference occurs when the equation (7) is implicit. In this case, the computation p_i requires solving the system of linear equations where the vector p_i appears on the left and right-hand sides of (11). Similarly, the matrix N_{ij} appears on both sides in (8). Therefore, we cannot define N_{ij} and p_i sequentially. We have to solve all linear algebraic equations (8) or (11) simultaneously.

Let us introduce the so-called z-graph and p-graph. Both these directed graphs have k nodes, each node is numbered from 1 to k. Nodes are connected by some of possible edges, which are directed by arrows. In z-graph at each i–th node we have edges coming out of all nodes whose numbers belong to the index set Q_i. Similarly in each i-th node of p-graph we have edges coming from the nodes whose numbers belong to $\bar{Q}_i$. This way in z-graph at each i-th node we define all vectors $z_j \in Z_i$ which are required in (10) to calculate z_i (input data). Analogously in p-graph at each i-th node, the incoming edges determine all vectors p_j, $j \in \bar{Q}_i$ which are required to calculate p_i in (11). If $i \in Q_i$, then we have a loop from node to itself. Assume that i, j belong to D, then $j \in Q_i$ if and only if $i \in \bar{Q}_j$. We obtain this property from the following functional dependency:

$$z_i = F(i, \ldots, z_j, \ldots).$$

Hence z-graph and p-graph are the same except the directions of edges which are, respectively, opposite. Both graphs can be called "informational graphs", because they illustrate the structure of dependencies between components of vectors z_i and p_i, respectively. Beside these graphs two computational graphs must be defined, which describe the sequence of computations. These graphs are apparent in explicit case (from left to right for computation z_i and opposite for p_i). In general case we confront loops, which may contain many nodes (subsystems of nonlinear equations) and computational graphs are not trivial. We should mention that the "computational graph" described in [12], coincides with z-graph defined above.

Suppose that at each step of process (7) we determine every state vector z_i with error ε_i. Thus, instead of (7) we use the following formula:

$$z_i = F(i, Z_i, U_i) + \varepsilon_i, \quad 1 \leq i \leq k. \tag{14}$$

The auxiliary function can be written as

$$E(z,u,\varepsilon) = W(z,u) + \sum_{j\in D}[F^T(j,Z_j,U_j) + \varepsilon_j^T]p_j.$$

The both vectors z_i and p_i have the same dimensionality. If process (14) is explicit, then the vector ε_i is the machine precision of an arithmetic computation of the vector $F(i,Z_i,U_i)$. In the implicit case, the error norms $||\varepsilon_i||$ tend to be much larger and are mainly determined by the accuracy of the solution to the nonlinear equations (14). The solution is denoted by $z(u,\varepsilon)$ since it is a composite function of the vector u and the error vector $\varepsilon^T = [\varepsilon_1^T, \varepsilon_2^T, \ldots, \varepsilon_k^T]$. Therefore, we can write $\Omega(u,\varepsilon){=}W(z(u,\varepsilon),u)$.

Let us use the canonical equations (10)—(12). Consider ε_i in these expressions as a component of the vector u. We obtain exactly the same expressions as (11) and (12). Substituting $z(u,\varepsilon)$ in (11), we obtain $p(u,\varepsilon)$. Moreover, we find that the gradient of Ω with respect to ε_i is given by $p_i(u,\varepsilon) = d\Omega(u,\varepsilon)/d\varepsilon_i$. The vector p_i corresponding to the process (7) can be defined by

$$p_i = d\Omega(u, 0_n)/d\varepsilon_i. \tag{15}$$

Suppose that the vector u is given with an error and that, instead of u, we use $\bar{u} = u + \delta$. Then for process (14) we have

$$\Omega(\bar{u},\varepsilon) - \Omega(u,0) = \sum_{i=1}^{k}[\langle p_i(u,0),\varepsilon_i\rangle + \left\langle \frac{d\Omega(u,0)}{du_i}, \delta_i \right\rangle] + O(||\varepsilon||^2 + ||\delta||^2), \tag{16}$$

where the derivatives of Ω with respect to u_i are obtained from (12) and $\langle a,b\rangle$ denotes the inner product of vectors a and b. This estimate can be used instead of a laborious interval analysis. Theoretical and practical aspects of error estimation were investigated by Iri [13].

The formulas (15), (16), and similar expressions below have been used often in debugging various programs for computing vectors p_i. In problems where the differentiation of functions F is difficult, the codes for calculating the adjoint variables according to (11) are not correct at times, and we must use an alternative method for computing the sequence p_i. Here we describe one approach for finding p_i without solving system (11).

Suppose that the s-th component p_i^s of the vector p_i must be checked. Let $\bar{\varepsilon}$ denote a vector with all zero components except the s-th component, which is equal to the s-th component ε_i^s of the vector ε_i. The function $\Omega(u,\varepsilon)$ is differentiable with respect to ε at $\varepsilon = 0_n$. Therefore, owing to (16) we have

$$\Omega(u,\bar{\varepsilon}) = \Omega(u,0_n) + p_i^s\varepsilon_i^s + O((\varepsilon_i^s)^2).$$

Hence p_i^s is a derivative of Ω with respect to ε_i^s at the origin. Thus we can generate an approximation to p_i^s by computing perturbed process with different ε_i^s and using the divided differences:

$$p_i^s \approx [\Omega(u,\bar{\varepsilon}) - \Omega(u,0_n)]/\varepsilon_i^s.$$

Unfortunately, several computations of process (7) are necessary before a reasonable approximation to p_i^s can be obtained.

Throughout the reminder of this section we consider explicit system (7). Let us denote

$$W^k(z_1, z_2, ..., z_k, u) = W(z,u).$$

Using (10), we express the last state vector z_k via the previous vectors and substitute this expression in W. We now define a new function

$$W^{k-1}(z_1, z_2, ..., z_{k-1}, u) = W^k(z_1, z_2, ..., z_{k-1}, F(k, Z_k, U_k), u).$$

Analogously, by reverse recursion, we define a sequence of scalar functions W^i as

$$W^i(z_1, z_2, ..., z_i, u) = W^{i+1}(z_1, z_2, ..., z_i, F(i+1, Z_{i+1}, U_{i+1}), u), \quad 1 \le i \le k-1, \tag{17}$$

$$W^0(u) = W^1(F(1, Z_1, U_1), u).$$

Here we realized a process of sequential elimination of the state vectors z_i.

Theorem 2.1 *Suppose that*

1. *all mappings $F(i,Z_i,U_i)$, $\quad 1 \le i \le k$, and scalar function W are differentiable with respect to the vector z at the point z, which is found from (7);*
2. *the multistep process (7) is explicit.*

Then all vectors p_i are uniquely defined by (5) *and they are equal to*

$$p_i = \partial W^i(z_1, z_2, \dots, z_i, u)/\partial z_i, \quad 1 \le i \le k. \tag{18}$$

Proof. Consider the matrix $\Phi_z^T(z,u)$ for explicit process (7). Define $s \times s$ submatrix

$$Y_{ij} = \partial(F^T(j, Z_j, U_j) - z_j^T)/\partial z_i.$$

It is obvious that $Y_{ij} = 0_{ss}$ if $j < i$. If $i = j$, then $Y_{ij} = -I_s$, where I_q is the identity $q \times q$ matrix. Hence Y_{ij} is an upper triangular matrix and equation (5) can be rewritten as follows

$$\sum_{j=i}^{k} F_{z_i}^T(j, Z_j, U_j)p_j - p_i + W_{z_i}(z,u) = 0_s, \quad 1 \le i \le k. \tag{19}$$

The absolute value of determinant of the matrix $\Phi_z^T(z,u)$ is equal to one therefore this matrix is nondegenerate and we conclude that the linear algebraic equation (19) has unique solution for any explicit process (7). Taking into account the definition of the conjugate index set, we obtain that the system (19) can easily be solved by using backward formula (11) with terminal condition (13). Thus (11) gives us the solution of (19). It is similar to "back substitution", which is used in the method of Gaussian elimination for solving a system of linear equations.

Differentiating (17) with respect to z_i and using the chain rule, we obtain

$$\frac{\partial W^i(z_1, z_2, \dots, z_i, u)}{\partial z_i} = W_{z_i}(z,u) + \sum_{j \in \bar{Q}_i} F_{z_i}^T(j, Z_j, U_j)\frac{\partial W^j(z,u)}{\partial z_j}, \quad 1 \le i \le k. \tag{20}$$

From comparison expressions (11) and (20) we conclude that (18) holds. This completes the proof of Theorem 2.1. □

Below we consider various examples that illustrate the characteristic properties of the different approaches presented for evaluating gradients. For explicit processes in many cases the reverse mode of computation has an advantage over the forward mode.

3 DIFFERENTIATION OF ELEMENTARY FUNCTIONS

The expressions presented above cover a wide range of problems. In this section, we apply them to differentiating elementary functions.

The functions a^x $(a > 0)$, x^α, $\log_a x$ $(a > 0,\ a \neq 1)$, $\sin x$, $\cos x$, $\tan x$, $\cot x$, $\arcsin x$, $\arccos x$, $\arctan x$, $\operatorname{arccot} x$ are called *main elementary functions*. We assume that the codes for calculating the main elementary functions and their derivatives are stored electronically and the calculations are carried out exactly (or with machine precision). One can add new functions to the list of the main elementary functions, if necessary.

Assume that a real-valued function $f(u)$ is given explicitly. We say that a function $f(u)$ is *an elementary function* if it can be represented as a finite composition of main elementary functions and arithmetic operations.

Suppose that we have to calculate partial derivatives of a scalar-valued function $f(u)$, $u \in \mathbb{R}^r$ with respect to all components u_i, $1 \le i \le r$. The function f is assumed to be a differentiable elementary function. Therefore, $f(u)$ can be defined by a sequential program. We introduce a new vector $z \in \mathbb{R}^k$ of intermediate variables. The evaluation of $f(u)$ is now carried out as a k-step computational process:

$$z_1 = F(1, Z_1, U_1), \quad z_2 = F(2, Z_2, U_2), \dots, \quad z_k = F(k, Z_k, U_k), \tag{21}$$

where all $z_j \in \mathbb{R}^1$, $z_k = f(u)$; Z_i and U_i are sets of some components of the vectors z and u, respectively; Z_1 is empty. The sets Z_i consist of the already computed quantities z_j with $j < i$. In other words, f is the composition of basic operations whose derivatives are assumed to be computable for all arguments of interest.

We introduce scalars $p_i \in \mathbb{R}^1$ and define the function

$$E(z,u,p) = W(z) + \sum_{i=1}^{k} F(i, Z_i, U_i)p_i.$$

In (10)–(12) we set $W(z) = z_k$, therefore, (13) yields $p_k = 1$. Using (11) and (12), we find the gradient of $f(u)$ in reverse mode. In this way we obtain the following expression for FAD:

$$p_i = \sum_{q \in \bar{Q}_i} F_{z_i}^T(q, Z_q, U_q)p_q, \quad \partial f(u)/\partial u_i = \sum_{q \in \bar{K}_i} F_{u_i}^T(q, Z_q, U_q)p_q. \tag{22}$$

Many publications analyze various algorithms for automatic differentiation from the point of view of the algebraic complexity of the computation, i.e., the total number of arithmetic operations required to compute a function and its partial derivatives. For results in this field, we refer the interested readers to [2, 9, 10, 11, 12, 13, 14, 15] and the relevant references cited therein.

Let T_0 denote the total time required to calculate the value of the underlying function $f(u)$. Let T_g denote the additional time required for computing all partial derivatives $\partial f(u)/\partial u_i$, $1 \leq i \leq r$.

Theorem 3.1 *Suppose that*

1. *$f(u)$ is an elementary scalar-valued differentiable function of the vector $u \in \mathbb{R}^r$;*
2. *the time for computing the derivative of each main elementary function is less than twice the time required to evaluation the main elementary function itself;*
3. *the time required for memory processing and for execution of the assignment operator is negligible.*

If the formulas (22) for fast automatic differentiation of $f(u)$ are used, then the gradient f_u is exact and the ratio $R = T_g/T_0$ is bounded above by 3.

For comparison, recall that if we approximate the derivatives by divided differences, this ratio is $R = r$, and the gradient is not exact for any nonlinear scalar function f. The Theorem 3.1 is proved on the basis of the approach developed in [10, 11, 12, 13, 14].

Let us consider a simple example of an elementary function

$$f(u) = e^{u_1+u_2} + \sin(u_1+u_2)^2.$$

The evaluation of this function is represented as the sequence of computations

$$\begin{aligned} z_1 &= u_1 + u_2, \\ z_2 &= e^{z_1}, \\ z_3 &= (z_1)^2, \\ z_4 &= \sin z_3, \\ z_5 &= z_2 + z_4. \end{aligned}$$

We introduce the Lagrangian multipliers and function E:

$$E(z,u,p) = z_5 + p_1(u_1+u_2) + p_2 e^{z_1} + p_3(z_1)^2 + p_4 \sin z_3 + p_5(z_2+z_4).$$

The reverse computation of multipliers is represented by

$$\begin{aligned} p_1 &= p_2 e^{z_1} + 2p_3 z_1, \\ p_2 &= p_5, \\ p_3 &= p_4 \cos z_3, \\ p_4 &= p_5, \\ p_5 &= 1. \end{aligned}$$

Solving this linear system, we obtain

$$p_1 = e^{z_1} + 2z_1 \cos z_3, \quad p_2 = p_4 = p_5 = 1, \quad p_3 = \cos z_3.$$

Therefore, (12) yields

$$df(u)/du_1 = df(u)/du_2 = e^{u_1+u_2} + 2(u_1+u_2)cos(u_1+u_2)^2.$$

In reverse direction we define the sequence of functions W^i and their derivatives:

$$\begin{aligned} W^5 &= z_5, & \partial W^5/\partial z_5 &= 1, \\ W^4 &= z_2 + z_4, & \partial W^4/\partial z_4 &= 1, \\ W^3 &= z_2 + \sin z_3, & \partial W^3/\partial z_3 &= \cos z_3, \\ W^2 &= z_2 + \sin(z_1)^2, & \partial W^2/\partial z_2 &= 1, \\ W^1 &= e^{z_1} + \sin(z_1)^2, & \partial W^1/\partial z_1 &= e^{z_1} + 2z_1 \cos z_3, \\ W^0 &= f(u). \end{aligned}$$

It is obvious that (18) holds.

4 DERIVATIVES WITH RESPECT TO INITIAL CONDITIONS

The expressions for the forward and reverse modes of differentiation should not be considered as a stand-alone topic in mathematical analysis. In this section we present similar results which were obtained in the theory of ordinary differential equations. These results can be derived from expressions given in Section 2. We briefly illustrate how this can be done. To compare the forward and reverse modes of differentiation, we consider the simplest problem in which we have to differentiate a function $W(z),\ z \in \mathbb{R}^n$ with respect to initial condition of a solution of ordinary differential equation.

Let the process be described by the following system of ordinary differential equations:

$$dz/dt = f(z), \quad T_1 \leq t \leq T_2. \tag{23}$$

The solution of (23) is a function $z(t, T_1, z_1)$, with initial condition $z(T_1, T_1, z_1) = z_1$. Assume that for each z the right-hand side of (23) is a continuously differentiable function of z and satisfies the known existence and uniqueness conditions for solution of Cauchy problem for the equation (23). In this case for all $T_1 \leq s \leq t \leq T_2$ we have

$$z(t, T_1, z_1) = z(t, s, z(s, T_1, z_1)).$$

We will find the derivative of the composite function $\Omega(z_1) = W(z(T_2, T_1, z_1))$.

By applying Euler numerical integration method, we obtain the following discrete approximation of continuous system (23)

$$z_1 = u, \quad z_i = z_{i-1} + hf(z_{i-1}), \quad 2 \leq i \leq k,$$

where $h = (T_2 - T_1)/(k-1)$.

In discrete case $\Omega(z_1) = W(z_k)$. Denote $N_i = dz_i^T/du,\ 1 \leq i \leq k,\ N_i \in \mathbb{R}^{n \times n}$. Then, using (8), we obtain

$$N_1 = I_n, \quad N_i = N_{i-1} + hN_{i-1}f_z^T(z_i), \quad 2 \leq i \leq k. \tag{24}$$

. Introducing the auxiliary function

$$E(z, u, p) = W(z_k) + p_1^T u + \sum_{i=2}^{k} p_i^T (z_{i-1} + hf(z_{i-1})),$$

and using (11) and (13), we obtain the difference equations for vectors $p_i \in \mathbb{R}^n$:

$$p_i = p_{i+1} + hf_{z_i}^T(z_i)p_{i+1}, \quad p_k = W_{z_k}(z_k), \quad 1 \leq i \leq k-1. \tag{25}$$

According to Theorem 2.1 the vector p is uniquely defined by (25). The desired gradient can be computed by either of the two formulas:

$$d\Omega(z_1)/dz_1 = N_k W_{z_k}(z_k), \quad d\Omega(z_1)/dz_1 = p_1.$$

For continuous process described by (23) we define a $n \times n$ matrix-function

$$N(t) = \frac{dz^T(t, T_1, z_1)}{dz_1}$$

with the initial condition $N(T_1) = I_n$. In the limit, as $h \to 0$ and $k \to \infty$, we find from (24) that the resulting continuous trajectory is described by the following matrix differential equation:

$$\frac{dN(t)}{dt} = N(t)f_z^T(z(t, T_1, z_1)). \tag{26}$$

Using (25), it easy to show that in the limit as $h \to 0$ and $k \to \infty$, the adjoint (or costate) vector $p(t)$ satisfies the following vector differential equation:

$$\frac{dp(t)}{dt} = -f_z^T(z(t, T_1, z_1))p(t), \tag{27}$$

here we defined a n-dimensional vector-function $p(t)$ with the terminal condition

$$p(T_2) = W_z(z(T_2, T_1, z_1)). \tag{28}$$

The equations (26) and (27) appear in publications devoted to the theory of ordinary differential equations [4]. Equation (26) can be found by differentiating both sides of equation (23) with respect to z_1, and using the chain rule.

As in the discrete case above, the gradient of the composite function Ω can be computed in two ways. The first computational process (from left to right) is based on the integration of differential equations (23) and (26) forward in time from $t = T_1$ to $t = T_2$ for the given z_1 and initial condition $N(T_1) = I_n$. The other one is based on the integration of differential equation (27) with the terminal condition (28) backward in time from $t = T_2$ to $t = T_1$ (from right to left).

The gradient of Ω can be computed, respectively, by one of the two formulas:

$$d\Omega(z_1)/dz_1 = N(T_2)W_z(z(T_2, T_1, z_1)), \quad d\Omega(z_1)/dz_1 = p(T_1).$$

The last expression was obtained in [13]. Both expressions give exactly the same result, but in the first case, we have to integrate the matrix system (26), which consists of n^2 scalar differential equations. In the second case, we integrate vector system (27) which consists of only n scalar differential equations. Thus the second approach will be n times less costly than the first one. The formula (27) can be used for numerical solution of two-point boundary-value problems. The technique is based on the multiple back-and-forth shooting method which transforms a given boundary-value problem into a sequence of initial-value problems. The method involves forward integration of (23) and backward integration of (27).

The vector $p(T_1)$ is a total derivative of the function W with respect to initial condition z_1. We can take as an initial condition for system (23) at time t the state vector $z(t, t, z(t, T_1, z_1))$. The solution of (23) initiating from this point will be the same as a solution $z(t, T_1, z_1)$ of (23) (because of uniqueness of the solutions passing through the point $z(t, t, z(t, T_1, z_1))$ at time t). The same function $p(t)$ will be found from integration the adjoint system (27) with the terminal condition (28). Consequently we obtain that the total derivative of W with respect to new initial condition is equal to

$$p(t) = \frac{dW(z(T_2, t, z(t, T_1, z_1)))}{dz(t, T_1, z_1)}.$$

This result is similar to (15) and (18) but in contrast to general case considered in the second section it gives us a very simple, clear explanation of vector-function $p(t)$ and it can be also used for debugging computational codes.

We should mention the less common case where forward differentiation is less time consuming than backward differentiation. This case arises when we need to find the gradients of m functions $\Omega^i(z_1) = B^i(z(T_2, T_1, z_1))$, where $1 \le i \le m$. We define $N(T_2)$ from (26), and all gradients are found as follows:

$$d\Omega^i(z_1)/dz_1 = N(T_2)B^i_z(z(T_2, T_1, z_1)).$$

Therefore, if $m > n$, then the forward mode is more efficient than the reverse mode.

5 THE OPTIMAL CONTROL PROBLEM

Optimal control theory has been formalized as an extension of the calculus of variations. It has many successful applications in various disciplines ranging from mathematics and engineering to economics, social and management sciences. Many numerical methods for solving optimal control problems have been proposed, and the research continues, especially in the field of nonlinear problems.

The basic optimal control problem can be described as follows. Let a process be governed by a system of ordinary differential equations:

$$dz/dt = f(t, z, u, \xi), \quad T_1 \le t \le T_2, \quad z(T_1, T_1, z_1) = z_1, \tag{29}$$

where the state vector $z \in \mathbb{R}^n$, the control u is an arbitrary piecewise continuous function of t having its values in U. The feasible set U is a given compact subset in the space $\mathbb{R}^r$. The vector of design parameters is $\xi \in V \subset \mathbb{R}^s$. As a rule, the scalars T_1, T_2 and vector z_1 are fixed. If T_1, T_2, z_1 must be optimized then we include them into vector ξ.

The problem is to find a control function $u(t) \in U$ and a vector of design parameters $\xi \in V$ that minimize the cost functional $W(z(T_2, T_1, z_1), \xi)$, subject to mixed constraints on state, control and vector of design parameters:

$$g(t, z(t), u(t), \xi) = 0, \quad q(t, z(t), u(t), \xi) \le 0, \quad T_1 \le t \le T_2.$$

As a rule, this problem is reduced to a mathematical programming problem by using the control parameterization technique. As an example, we consider here the simplest discretized version of (29), which is given by the Euler formula

$$z_i = z_{i-1} + h_{i-1} f(t_{i-1}, z_{i-1}, u_{i-1}, \xi) = F(t_{i-1}, z_{i-1}, u_{i-1}, \xi), \tag{30}$$

where $\sum_{i=2}^{k} h_{i-1} = T_2 - T_1$, $0 < h_i$, $t_1 = T_1$, $t_k = T_2$, $t_i = t_{i-1} + h_{i-1}$, $2 \le i \le k$.

Thus we approximate the control by a piecewise constant function. We take into account the mixed constraints at each grid point:

$$g(t_i, z_i, u_i, \xi) = 0, \quad q(t_i, z_i, u_i, \xi) \le 0, \quad 1 \le i \le k. \tag{31}$$

Now the objective function is $W(z_k, \xi)$ and the auxiliary function is expressed as

$$E(z, u, p, \xi) = W(z_k, \xi) + \sum_{i=2}^{k} F^T(t_{i-1}, z_{i-1}, u_{i-1}, \xi) p_i.$$

Discretizing control and constraints, we arrive at the following parameterization-discretization scheme for an approximate solution of optimal control problem.

Minimize $W(z_k, \xi)$ with respect to $u_i \in U$, $\quad 1 \le i \le k$, and $\xi \in V$, subject to mixed constraints (31).

In order to apply NLP solvers to this problem we must have an efficient algorithm for computing the first order derivatives of the objective and the constraint functions. Applying the results of Section 2, we find that all vectors $p_i \in \mathbb{R}^n$ and the derivatives of $\Omega(u, \xi) = W(z_k, \xi)$ can be calculated from

$$\begin{aligned}
p_i &= p_{i+1} + h_i f_{z_i}^T(t_i, z_i, u_i, \xi) p_{i+1}, \quad 1 \le i \le k-1, \qquad p_k = W_{z_k}(z_k, \xi), \\
d\Omega/du_i &= h_i f_{u_i}^T(t_i, z_i, u_i, \xi) p_{i+1}, \qquad 1 \le i \le k-1, \qquad d\Omega/du_k = 0, \\
d\Omega/d\xi &= W_\xi(z_k, \xi) + \sum_{i=2}^{k} F_\xi^T(t_{i-1}, z_{i-1}, u_{i-1}, \xi) p_i, \\
p_i &= dW(z_k, \xi)/dz_i, \quad 1 \le i \le k-1.
\end{aligned} \tag{32}$$

The finite-dimensional approximate problems are solved by standard or adapted nonlinear programming methods (penalty function method, modified Lagrangian, gradient projection method, linearization, interior point techniques, etc.). The gradient methods have been successfully implemented and have been found to be more effective and robust than derivative-free methods. Second derivative methods are most attractive, but suffer from high dimensionality.

To solve the discretized problem by standard nonlinear programming methods we introduce additional functions (for example, a penalty function, the Lagrangian, the modified Lagrangian and so on). According to (11)— (12), the expressions for computing their derivatives will be similar to (32) if instead of W we substitute these functions and take into account all nonzero derivatives W_{u_i}, W_ξ and W_{z_i}.

If we use a differentiable exterior penalty function with penalty coefficient τ, then the function E is defined as follows:

$$\begin{aligned}
E = W(z_k, \xi) &+ \sum_{i=2}^{k} F^T(t_{i-1}, z_{i-1}, u_{i-1}, \xi) p_i + \\
&+ \tau \sum_{i=1}^{k} [\|g(t_i, z_i, u_i, \xi)\|^2 + \|q_+(t_i, z_i, u_i, \xi)\|^2],
\end{aligned}$$

where a_+ denotes the vector in $\mathbb{R}^l$ with components

$$(a_+)^i = \max[a^i, 0], \quad 1 \le i \le l.$$

The expressions for computing gradients of the objective and constraints are derived via the adjoint system for the Euler and Runge-Kutta discretization schemes in [7]. These expressions and similar results for second derivatives were used for solving the optimal control problem with mixed constraints by gradient and Newton's methods. The implementation of expressions (32), a short description of an optimal control package, and some numerical illustrative examples are given in [7, 8].

In the limit as $h_i \to 0$ and $k \to \infty$, we find from (32) that the function $p(t)$ satisfies the following differential equation:

$$\dot{p} = -f_z^T(t, z, u, \xi) p, \quad p(T_2) = W_z(z(T_2, T_1, z_1), \xi). \tag{33}$$

This is the so-called costate (or adjoint, or conjugate) equation, which is used in Pontryagin maximum principle [18]. The adjoint system is integrated backward in time, from the $t = T_2$ to $t = T_1$.

If system (29) is stiff, we have to use an implicit integration scheme [20]. For example, the application of the implicit Euler formula leads to

$$z_i = z_{i-1} + h_i f(t_i, z_i, u_i, \xi), \quad 2 \le i \le k. \tag{34}$$

Although we have to solve a system of nonlinear equations at each step in this case, we can take a much bigger step size h_i in (34) than in (30). We obtain the following discrete costate equations from (11):

$$p_i = p_{i+1} + h_i f_{z_i}^T(t_i, z_i, u_i, \xi) p_i, \qquad 2 \le i \le k-1,$$

$$p_k = W_{z_k}(z_k, \xi) + h_k f_{z_k}^T(t_k, z_k, u_k, \xi) p_k.$$

Hence all vectors p_i are found from implicit linear algebraic system. The gradient expressions become

$$d\Omega/du_i = h_i f_{u_i}^T(t_i, z_i, u_i, \xi) p_i, \quad 2 \le i \le k, \quad d\Omega/du_1 = 0.$$

In the limit ($h_i \to 0$, $k \to \infty$) we obtain the same expressions as given in (33).

In some publications, the gradients are found from necessary optimality conditions by discretizing the initial and costate systems. In this case, some errors may arise. Indeed, if we simultaneously discretize the system of ordinary differential equations (29) and (33) using the Euler scheme, then we obtain (30) and

$$p_{i+1} = p_i - h_i f_{z_i}^T(t_i, z_i, u_i, \xi) p_i, \quad 1 \le i \le k-1,$$

$$d\Omega/du_i = h_i f_{u_i}^T(t_i, z_i, u_i, \xi) p_i.$$

These expressions do not coincide with (32) and therefore the gradient based on this approach is not correct. If h_i is rather small then the difference between this expression and exact expression (32) is $O(h_i^2)$. But the error in the gradient calculation is not desirable when we use sensitive minimization algorithms (for example conjugate gradients) or when the step size h_i is not small enough.

We emphasize the following important conclusion. When we deal with optimal control problems, the discretization of the costate equation must correspond to the integration scheme of the initial system. It should not be performed independently. The same is required in the case of more complicated processes described by partial differential equations. However, these requirements are not as obvious there as in the case of ordinary differential equations.

6 THE OPTIMAL CONTROL PROBLEM OF A PARABOLIC SYSTEM

Optimal control theory for distributed parameter systems has been extensively studied in [17, 21, 22, 23] and in many others research publications. In this and in the next sections we will show that our approach gives us a new methodology for finding exact gradients in complicated controlled systems governed by partial differential equations.

We discretize the infinite dimensional optimization problem to obtain an approximate finite dimensional nonlinear programming problem. The determination of the functional gradient is done using the adjoint variable formulation.

Consider, as an example, the second order parabolic heat equation

$$\frac{\partial z(x,t)}{\partial t} = a^2 \frac{\partial^2 z(x,t)}{\partial x^2} + u(x,t), \quad 0 < x < l, \quad 0 < t < T, \tag{35}$$

where $z(x,t)$ is the temperature at time t at a point x and $u(x,t)$ is a distributed control.

The initial and boundary conditions are given by

$$z(x,0) = \varphi(x), \quad 0 \le x \le l, \tag{36}$$

$$\frac{\partial z(0,t)}{\partial x} = 0, \quad \frac{\partial z(l,t)}{\partial x} = \nu[g(t) - z(l,t)], \quad 0 < t \le T, \tag{37}$$

where $g(t)$ is a boundary control.

The problem is to find control functions $u(x,t)$ and $g(t)$ that minimize the cost functional

$$W = \int_0^l \Psi(z(s,T))ds, \tag{38}$$

where Ψ is continuously differentiable with respect to its argument.

For given control functions $u(x,t)$ and $g(t)$, we solve equation (35) with conditions (36) and (37), and then substitute this solution into (38) to evaluate W. This value is a composite function of u and g. Denote it by $\Omega(u,g)$.

Since the optimal control cannot be obtained as an analytic solution of the necessary and sufficient optimality conditions, we attempt to find it numerically by minimizing Ω via a descent algorithm. We are thus faced with computing the gradient of the cost functional for which we apply the expressions given in Section 2. The problem is discretized by a finite difference approximation scheme. For the simplicity, we use a uniform grid and denote

$$x_i = i\Delta x, \quad t_j = j\Delta t, \quad i = 0,\dots,k, \quad j = 0,\dots,m,$$

$$\Delta x = l/k, \quad \Delta t = t/m, \quad z_i^j = z(i\Delta x, j\Delta t), \quad u_i^j = u(i\Delta x, j\Delta t),$$
$$\varphi_i = \varphi(i\Delta x), \quad g^j = g(j\Delta t), \quad i = 0,\dots,k, \quad j = 0,\dots,m.$$

Let us discretize the parabolic equation (35) with an explicit forward Euler scheme in time, a centered scheme in space and use the simplest discretization in the vicinity of boundaries. Then the cost functional (38), the differential equation (35) and the conditions (36), (37) are replaced by

$$\tilde{W} = \Delta x \sum_{i=0}^{k} \alpha_i \Psi(z_i^m), \tag{39}$$

$$z_i^j = \begin{cases} (1-2\lambda)z_i^{j-1} + \lambda(z_{i-1}^{j-1} + z_{i+1}^{j-1}) + \Delta t u_i^{j-1}, & 1 \le i \le k-1, & 1 \le j \le m, \\ z_1^j, & i = 0, & 1 \le j \le m, \\ \mu z_{k-1}^j + \mu\nu\Delta x g^j, & i = k, & 1 \le j \le m, \\ \varphi_i, & 0 \le i \le k, & j = 0, \end{cases} \tag{40}$$

where α_i are the quadrature coefficients, $\lambda = a^2\Delta t/(\Delta x)^2$, $\mu = 1/(1+\nu\Delta x)$.

We introduce the adjoint variables p_i^j and the auxiliary function

$$E = \tilde{W} + \sum_{i=1}^{k-1}\sum_{j=1}^{m}[(1-2\lambda)z_i^{j-1} + \lambda(z_{i-1}^{j-1} + z_{i+1}^{j-1}) + \Delta t u_i^{j-1}]p_i^j +$$

$$+ \sum_{j=1}^{m}[z_1^j p_0^j + \mu(z_{k-1}^j + \nu\Delta x g^j)p_k^j] + \sum_{i=0}^{k}\varphi_i p_i^0.$$

Applying formula (11) we obtain

$$p_i^j = \begin{cases} (1-2\lambda)p_i^{j+1} + \lambda(p_{i-1}^{j+1} + p_{i+1}^{j+1}), & 2 \le i \le k-2, & 0 \le j \le m-1, \\ (1-2\lambda)p_1^{j+1} + p_0^j + \lambda p_2^{j+1}, & i = 1, & 1 \le j \le m-1, \\ (1-2\lambda)p_{k-1}^{j+1} + \mu p_k^j + \lambda p_{k-2}^{j+1}, & i = k-1, & 1 \le j \le m-1, \\ \alpha_i \Delta x \Psi_{z_i^m} + p_0^m \delta_i^1 + \mu p_k^m \delta_i^{k-1}, & 0 \le i \le k, & j = m, \\ (1-2\lambda)p_i^1 + \lambda p_{i+1}^1, & i = 1, i = k-1, & j = 0, \\ \lambda p_1^{j+1}, & i = 0, & 0 \le j \le m-1, \\ \lambda p_{k-1}^{j+1}, & i = k, & 0 \le j \le m-1, \end{cases}$$

where $\delta_\alpha^\beta = 1$ if $\alpha = \beta$, else $\delta_\alpha^\beta = 0$. Then, using (12) we find the derivatives

$$\begin{array}{lll} d\Omega/du_i^j = \Delta t p_i^{j+1}, & 1 \le i \le k-1, & 0 \le j \le m-1, \\ d\Omega/du_0^j = d\Omega/du_k^j = 0, & & 0 \le j \le m-1, \\ d\Omega/du_i^m = 0, & 0 \le i \le k, & \\ d\Omega/dg^j = \mu\nu\Delta x p_k^j, & & 1 \le j \le m. \end{array} \tag{41}$$

According to [19], the discrete approximation (40) is stable only if $\lambda \le 1/2$. We can replace it with the more complicated implicit scheme which is stable for all λ:

$$z_i^{j+1} = z_i^j + \lambda(z_{i-1}^{j+1} - 2z_i^{j+1} + z_{i+1}^{j+1}) + \Delta t u_i^{j+1}.$$

In the interior region, the vector p is now defined by the difference equation

$$p_i^j = p_i^{j+1} + \lambda(p_{i+1}^j - 2p_i^j + p_{i-1}^j).$$

If we let $k \to \infty$, $\Delta t \to 0$, $\Delta x \to 0$, then in both cases we find that the function $p(x,t)$ satisfies the following conditions:

$$\begin{aligned} &\frac{\partial p(x,t)}{\partial t} + a^2 \frac{\partial^2 p(x,t)}{\partial x^2} = 0, \quad 0 < x < l, \quad 0 < t < T, \\ &p(x,t) = \Psi_z(z(x,T)), \qquad\qquad\qquad 0 \le x \le l, \\ &\frac{\partial p(0,t)}{\partial x} = 0, \quad \frac{\partial p(l,t)}{\partial x} + \nu p(l,t) = 0, \qquad 0 < t < T. \end{aligned} \tag{42}$$

Discrete approximations of all these conditions were obtained on the basis of (11) and, therefore, (41) gives us the exact gradients for the discretized process (40). Equation (42) is conjugate (or adjoint) to (35).

The gradients of the cost functional for the continuous problem are given by

$$d\Omega/du(x,t) = p(x,t), \quad d\Omega/dg(t) = \nu a^2 p(l,t). \tag{43}$$

Using the expressions widely used in classical calculus of variations, we can rewrite (43) in the following standard form:

$$\delta\Omega = \int_0^T \int_0^l p(x,t)\delta u(x,t)dxdt + \nu a^2 \int_0^T p(l,t)\delta g(t)dt.$$

This clarifies notation (43). These functional derivatives were obtained in [23] by calculus of variations.

7 THE OPTIMAL CONTROL PROBLEM OF A HYPERBOLIC SYSTEM

Consider now a dynamical system described by the linear hyperbolic partial differential equation

$$\frac{\partial^2 z(x,t)}{\partial t^2} = a^2 \frac{\partial^2 z(x,t)}{\partial x^2} + u(x,t), \quad 0 < x < l, \quad 0 < t < T, \tag{44}$$

with the initial and boundary conditions

$$z(x,0) = \varphi^1(x), \quad z_t(x,0) = \varphi^2(x), \quad 0 \le x \le l,$$

$$z_x(0,t) = g_1(t), \quad z_x(l,t) = g_2(t), \quad 0 < t \le T.$$

Controls $u(x,t)$, $g_1(t)$, $g_2(t)$ are chosen to minimize the functional:

$$W = \int_0^l \Psi(z(s,T), z_t(s,T))ds,$$

where the function Ψ is continuously differentiable with respect to its arguments.

Replacing the continuous optimal control problem by its discrete version, we construct an approximation

$$\tilde{W} = \Delta x \sum_{i=0}^{k} \alpha_i \Psi(z_i^m, (z_i^m - z_i^{m-1})/\Delta t),$$

$$z_i^j = \begin{cases} 2(1-\lambda)z_i^{j-1} - z_i^{j-2} + \lambda(z_{i+1}^{j-1} + z_{i-1}^{j-1}) + \Delta t^2 u_i^{j-1}, & 1 \le i \le k-1, \quad 2 \le j \le m, \\ z_1^j - \Delta x g_1^j, & i = 0, \quad 1 \le j \le m, \\ z_{k-1}^j + \Delta x g_2^j, & i = k, \quad 1 \le j \le m, \\ \varphi_i^1, & 0 \le i \le k, \quad j = 0, \\ z_i^0 + \Delta t \varphi_i^2, & 1 \le i \le k-1, \quad j = 1, \end{cases}$$

where $\lambda = a^2(\Delta t)^2/(\Delta x)^2$, $\varphi_i^1 = \varphi^1(i\Delta x)$, $\varphi_i^2 = \varphi^2(i\Delta x)$, $g_1^j = g_1(j\Delta t)$, $g_2^j = g_2(j\Delta t)$.

Applying here the results of Section 2, as in the previous section, we define

$$E = \tilde{W} + \sum_{i=1}^{k-1} \sum_{j=2}^{m} [2(1-\lambda)z_i^{j-1} - z_i^{j-2} + \lambda(z_{i+1}^{j-1} + z_{i-1}^{j-1}) + \Delta t^2 u_i^{j-1}]p_i^j +$$

$$+\sum_{j=1}^{m}[(z_1^j-\Delta x g_1^j)p_0^j+(z_{k-1}^j+\Delta x g_2^j)p_k^j]+\sum_{i=0}^{k}\varphi_i^1 p_i^0+\sum_{i=1}^{k-1}(z_i^0+\Delta t\varphi_i^2)p_i^1.$$

The adjoint variables are found from the following discrete algebraic difference equations

$$p_i^j=\begin{cases} 2(1-\lambda)p_i^{j+1}-p_i^{j+2}+\lambda(p_{i-1}^{j+1}+p_{i+1}^{j+1}), & 2\le i\le k-2, \quad 1\le j\le m-2,\\ 2(1-\lambda)p_1^{j+1}-p_1^{j+2}+\lambda p_2^{j+1}+p_0^j, & i=1, \quad 1\le j\le m-2,\\ 2(1-\lambda)p_{k-1}^{j+1}-p_{k-1}^{j+2}+\lambda p_{k-2}^{j+1}+p_k^j, & i=k-1, \quad 1\le j\le m-2,\\ \lambda p_1^{j+1}\delta_i^0+\lambda p_{k-1}^{j+1}\delta_i^k, & i=0,\ i=k, \quad 1\le j\le m-2,\\ \tilde W_{z_i^j}+2(1-\lambda)p_i^m+\lambda(p_{i-1}^m+p_{i+1}^m), & 2\le i\le k-2, \quad j=m-1,\\ \tilde W_{z_1^j}+p_0^j+2(1-\lambda)p_1^m+\lambda p_2^m, & i=1, \quad j=m-1,\\ \tilde W_{z_i^j}+p_k^j+2(1-\lambda)p_i^m+\lambda p_{i-1}^m, & i=k-1, \quad j=m-1,\\ \tilde W_{z_i^j}+\lambda p_1^m\delta_i^0+\lambda p_{k-1}^m\delta_i^k, & i=0,\ i=k, \quad j=m-1,\\ \tilde W_{z_i^m}+p_0^m\delta_i^1+p_k^m\delta_i^{k-1}, & 0\le i\le k, \quad j=m,\\ p_i^1-p_i^2, & 1\le i\le k-1, \quad j=0,\\ 0, & i=0,\ i=k, \quad j=0.\end{cases}$$

The required derivatives are defined by

$$\begin{array}{lll} d\Omega/du_i^j=\Delta t^2 p_i^{j+1}, & 1\le i\le k-1, & 1\le j\le m-1,\\ d\Omega/du_0^j=d\Omega/du_k^j=0, & & 0\le j\le m-1,\\ d\Omega/du_i^0=d\Omega/du_i^m=0, & 0\le i\le k, & \\ d\Omega/dg_1^j=-\Delta x p_0^j, \quad d\Omega/dg_2^j=\Delta x p_k^j, & & 1\le j\le m,\\ d\Omega/dg_1^0=d\Omega/dg_2^0=0. & & \end{array}$$

If we let $k\to\infty$, $\Delta t\to 0$, $\Delta x\to 0$, then the conjugate equation and boundary conditions are given by

$$\frac{\partial^2 p(x,t)}{\partial t^2}=a^2\frac{\partial^2 p(x,t)}{\partial x^2}, \quad 0<x<l, \quad 0<t<T, \tag{45}$$

$$\frac{\partial p(0,t)}{\partial x}=0, \quad \frac{\partial p(l,t)}{\partial x}=0, \qquad 0<t<T.$$

As earlier, we have automatically obtained the discrete scheme of the adjoint equation (45) and initial and boundary conditions which yield an exact gradient. Once the exact derivatives of the discretized cost functional are determined, the minimization can be performed by using any available gradient algorithm. Thus, we don't see any necessity to choose a special finite-dimensional approximation of (44) and other conditions in order to get "a good discretized gradient", which is the main aim of many publications including the cited paper [16].

8 CONCLUSION

We have described a technique that provides a unified tool for computing the gradients for complex systems defined by explicit and implicit expressions. The proposed approach requires only a solution of the linear costate system after the cost function is evaluated. The approach provides an opportunity to evaluate relatively easily the exact gradient of the cost function for the chosen discrete approximation of a continuous process.

The technique used above is by no means restricted to the described problems. It can be applied to problems that involve other types of distributed parameter systems and can be used for sensitivity analysis.

In order to distill our technique we have applied the proposed approach to the problem considered in [16] and we have obtained some illustrative computational results. The corresponding paper has been accepted for publication by the *Journal on Computational Mathematics and Mathematical Physics* and it will appear there late in 1997.

In conclusion, I would like to agree with M. Iri who wrote in [12] to the effect that the automatic differentiation technique is a fundamental tool for the future computations in science and technology, and it is a most worthy area of investment in research and development.

Acknowledgements

The author would like to thank E. Spedicato, N. Alexandrov, A. Albou, N. Tzannetakis, K.L. Teo, V. Rehbock, V. Zubov for useful suggestions and comments. The author is deeply indebted to Professor Masao Iri, who passed on numerous papers and reports devoted to FAD.

References

[1] K.R. Aida-Zade and Y.G. Evtushenko, Fast automatic differentiation, *Mathematical Modeling*, **1**, 121-139 (1989) (in Russian).

[2] W. Baur and V. Strassen, The complexity of partial derivatives, *Theoretical Computer Sciences*, **22**, 317-320 (1983).

[3] D. Bertsekas, *Constrained optimization and Lagrange multiplier methods*, Academic Press, New York (1982).

[4] C. Caratheodory, *Variationsrechnung und partielle Differentialgleichungen erster Ordnung*, Band 1, Leipzig (1956).

[5] Y.G. Evtushenko and V.P. Mazouric, *Optimization Software*, Znanie, Moscow (1989) (in Russian).

[6] Y.G. Evtushenko, *Automatic differentiation viewed from optimal control theory*, in [9], 25-30 (1991).

[7] Y.G. Evtushenko, *Numerical Optimization Techniques*, Optimization Software, Inc., Publications Division, New York (1985).

[8] N.I. Grachev and Y.G. Evtushenko, A library of programs for solving optimal control problems, *U.S.S.R. Comput. Maths. Math. Phys.* (Pergamon Press), **19**, 99-119 (1980).

[9] A. Griewank, *On automatic differentiation*, in: M. Iri and K. Tanabe (Eds.), Mathematical Programming: Recent Developments and Applications, Kluwer Academic Publishers, 83-108 (1989).

[10] A. Griewank and G.F. Corliss, Eds., *Automatic Differentiation of Algorithms. Theory, Implementation and Application*, SIAM, Philadelphia (1991).

[11] A. Griewank, Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation, *Optimization Methods and Software*, **1**, 35-54 (1992).

[12] M. Iri, *History of Automatic differentiation and rounding error estimation*, in [9], 3-16 (1991).

[13] M. Iri, Simultaneous computation of functions, partial derivatives and estimates of rounding errors — Complexity and practicality, *Japan Journal of Applied Mathematics*, **1**, 223-252 (1984).

[14] M. Iri and K. Kubota, *Methods of fast automatic differentiation and applications*, Research memorandum RMI 87-02. Department of Mathematical Engineering and Instrumental Physics, Faculty of Engineering, University of Tokyo (1987).

[15] K. Kim, Y. Nesterov, V. Skokov, B. Cherkasskij, Efficient algorithm for differentiation and extremal problem, *Economy and Mathematical Methods*, **20**, 309-318 (1984) (in Russian).

[16] J.M. Lellouche, J.L. Devenon, I. Dekeyser, Boundary control of Burger's equation — A numerical approach, *Computers and Mathematics with Applications*, **28**(5), 33-44 (1994).

[17] G.I. Marchuk, *Conjugate equations and analysis of complex systems*, Science, Moscow (1992) (in Russian).

[18] L.S. Pontryagin, V.G. Boltyansky, R.V. Gamkrelidze, E.F. Mitshenko, *Mathematical Theory of Optimal Process*, Nauka, Moscow (1961).

[19] A.A. Samarskii, *Introduction to the Theory of Finite-difference Methods*, Nauka, Moscow (1971) (in Russian).

[20] H.J. Stetter, *Analysis of Discrimination Methods for Ordinary Differential Equations*, Springer-Verlag, Berlin, Heidelberg, New York (1973).

[21] K.L. Teo and Z.S. Wu, *Computation Methods for Optimizing Distributed Systems*, Academic Press, Inc (1984).

[22] K.L. Teo, C.J. Goh, K.H. Wong, *A Unified Computation Approach to Optimal Control Problems*, Longman Scientific & Technical, England (1991).

[23] F.P. Vasiliev, *Numerical Methods for Solving Optimization Problems*, Nauka, Moscow (1981) (in Russian).