K. V. KIM, IU. E. NESTEROV, V. A. SKOKOV,
AND B. V. CHERKASSKII
(USSR)

An Efficient Algorithm for Computing Derivatives and Extremal Problems*

1. Introduction

The development and use of methods for solving smooth extremal problems depend significantly both on the ability to organize the computation of total derivatives of the functions included in the problems' constraints and on the difficulty of or time taken by this process. The most important property of any optimizing method is the rank of the derivatives it requires. The higher it is, then, as a rule, the more quickly the method converges. At the same time, in choosing a method for solving a particular problem it is necessary first to establish whether it is possible to program formulae for derivatives of the necessary order and whether the difficulty of computing them yields a return by increasing the speed of convergence. Comparative evaluations of the difficulty of computing derivatives usually start from the following plausible relationships:

*Russian text © 1984 by "Nauka" Publishers. "Effektivnyi algoritm vychisleniia proizvodnykh i ekstremal'nye zaduchi," *Ekonomika i matematicheskie metody*, 1984, vol. 20, no. 2, pp. 309–318.

$$t(F')=O(t(F)n),$$
$$t(F'')=O(t(F)n^2), \tag{1}$$

where F is a real function of n variables; F' is a vector of the gradient of function F, F'' is a matrix of second derivatives of function F; and $t(\varphi)$ is the difficulty of computing magnitude φ.

For example, if a first-order method is chosen to solve a particular problem, then the investigator must write out, for the functions included in the problem's constraints, formulae for the first-order partial derivatives (which sometimes is by no means easy) and compile a program for computing them. In fact the computation time on the computer depends significantly on how efficiently the program has been written. Equations (1) only indicate the existing uncertainty surrounding the question of the possible efficiency of a program which computes the first-order partial derivatives of a general non-linear function.

The present article proposes a method for computing the derivatives for a fairly general class of non-linear functions of n variables, using equations expressing the difficulty of the computation in the form:

$$\begin{aligned} t(F')&=O(t(F)),\\ t(F''y)&=O(t(F)),\\ t(F'')&=O(t(F)n), \end{aligned} \tag{2}$$

where y is an arbitrary direction in R^n and the rest of the notation is as in (1). We note that this method, which is justified and discussed in Section 2, can easily be automated.

Section 3 discusses the changes which should be introduced into a qualitative representation of the difficulty of some optimization methods when account is taken of the change from equations (1) to (2).

The present article is not concerned with implementation aspects of automating the method. We merely note that the

automated version frees the user from the need to write the programs for computing the derivatives. These programs are generated automatically on the basis of a given algorithm for computing the functions. In the first place this method substantially simplifies the preliminaries for solving the problem and secondly it provides the user with very efficient programs—which satisfy equations (2)—for computing the derivatives. Automated differentiation techniques can be used for a very broad class of non-linear methods. Reference [1] contains some information about similar developments abroad.

2. The differentiation algorithm

We consider algorithms for computing the first and second derivatives of a non-linear function in n variables when the difficulty of doing so is given by (2). We begin with some formal constructions.

We take a set of basic operations Ω, consisting of real functions $f_\alpha(x_1, \ldots, x_{m\alpha})$, $\alpha \in A \subseteq (0, 1)$, such that f_α: $G_\alpha \to R$, where G_α is some subset of space R^{m_α}. We choose an arbitrary natural number $n \geqslant 1$. Let $\mathfrak{M} \equiv \mathfrak{M}(s) = \{\mathfrak{A}, J\}$, where $\mathfrak{A} = \{\alpha_k\}_{k=n+1}^{n+s}$, $\alpha_k \in A$, $J = \{I_k\}_{k=n+1}^{n+s}$, $I_k = \{j_1^k, \ldots, j_{m_{\alpha_k}}^k\}$, $s \geqslant 1$. We shall call set $\mathfrak{M}$ correct if for any k, $n+1 \leqslant k \leqslant n+s$ and any $j \in I_k$ inequality $j < k$ is satisfied.

With each correct set $\mathfrak{M} = \mathfrak{M}(s)$ we associate a function of n variables $\varphi_{\mathfrak{M}}(x)$, the value of which at point $x = (x_1, \ldots, x_n)$ is computed using a recursive sequence of numbers $\{y_k\}_{k=1}^{n+s}$, according to rules:

$$y_i = x_i, \quad i = 1, \ldots, n,$$

$$y_k = f_{\alpha_k}(y_{j_1^k}, \ldots, y_{j_{l_k}^k}), \quad k = n+1, \ldots, n+s, \tag{3}$$

where $\{\alpha_k\}_{k=n+1}^{n+s} \equiv \mathfrak{A}$, $l_k = m_{\alpha_k}$, $\{j_1^k, \ldots, j_{l_k}^k\} \equiv I_k$, $\{I_k\}_{k=n+1}^{n+s} \equiv J$, $\{\mathfrak{A}, J\} = \mathfrak{M}$. Here $\varphi_{\mathfrak{M}}(x) = y_{n+s}$. We assume that $\varphi_{\mathfrak{M}}(\cdot)$ is defined only at x such that for any k, $n+1 \leqslant k \leqslant n+s$ the inclusion $(y_{j_1^k}, \ldots, y_{j_{l_k}^k}) \in G_{\alpha_k}$ is satisfied. We shall say that at these points the function $\varphi_{\mathfrak{M}}(\cdot)$ is computable. We note that all functions which are normally encountered can be represented in form (3).

We denoted by Φ_n the set of n variables corresponding to all possible correct sets $\mathfrak{M}$. For each function $\varphi_{\mathfrak{M}} \in \Phi_n$, we can introduce the notion of a formal gradient. For this purpose we define the set of gradients of basic operations Ω' as follows: with each operation $f_\alpha \in \Omega$ we associate a vector function:

$$f_\alpha'(x_1, \ldots, x_{m_\alpha}) = (f_\alpha^1(x_1, \ldots, x_{m_\alpha}), \ldots, f_\alpha^{m_\alpha}(x_1, \ldots, x_{m_\alpha})),$$

where f_α': $G_\alpha' \to R^{m_\alpha}$, G_α' is an open set from R^{m_α}, $\alpha \in A$. We call vector $f_\alpha'(x)$ the formal gradient of operation f_α at point $x \in G_\alpha'$.

We choose an arbitrary point $x \in R^n$ and consider sequences of numbers $\{y_k\}_{k=1}^{n+s}$, $\{p_k\}_{k=1}^{n+s}$ of vectors $\{g_t\}_{k=n+1}^{n+s}$, defined by recurrent equations:

$$\begin{gathered} y_i = x_i, \quad i = 1, \ldots, n, \\ y_k = f_{\alpha_k}(y_{j_1^k}, \ldots, y_{j_{l_k}^k}), \\ g_k = f_{\alpha_k}'(y_{j_1^k}, \ldots, y_{j_{l_k}^k}), \quad k = n+1, \ldots, n+s, \\ p_{n+s} = 1, \\ p_q = \sum_{t \in \omega_q} p_t g_t^{i_{qt}}, \quad q = n+s-1, \ldots, 1, \end{gathered} \tag{4}$$

where α_k, l_k, j_i^k have the same sense as in (3); ω_q is the set of indexes corresponding to all operations f_α, $\alpha \in \mathfrak{A}$ in which y_q enters as an argument; $g_t^{i_{qt}}$ is a component of vector g_t corresponding to magnitude y_q. We say that a function $\varphi_{\mathfrak{M}} \in \Phi_n$ is

formally differentiable at point $x \in R^n$, if in (4) for any k, $n+1 \leqslant k \leqslant n+s$ the following inclusion is satisfied:

$$(y_{j_1}^k, \ldots, y_{j_{l_k}}^k) \in G_{\alpha_k} \cap G'_{\alpha_k}.$$

Here we shall call vector $\varphi_{\mathfrak{M}}'(x) = (p_1, \ldots, p_n)$ a formal gradient of function $\varphi_{\mathfrak{M}}$ at point x. The correctness of this definition is justified by the following theorem.

Theorem 1. Assume that the following conditions are fulfilled for any $\alpha \in \mathfrak{A}$: (1) function f_α is continuously differentiable in set G_α'; (2) set $G_\alpha \cap G_\alpha'$ is open; (3) for any $x \in G_\alpha \cap G_\alpha'$ vector $f_\alpha'(x)$ is the gradient of function $f_\alpha(\cdot)$ at point x. Then if function $\varphi_{\mathfrak{M}}(\cdot)$ from Φ_n is formally differentiable at point $\bar{x} \in R^n$, then it is differentiable at that point and magnitudes $p_1, \ldots, p_n$ are its partial derivatives at point $\bar{x}$.

Proof. It is enough to show that there is an open neighborhood $B(\bar{x})$ of point $\bar{x}$ such that function $\varphi_{\mathfrak{M}}(\cdot)$ is computable at any $v \in B(\bar{x})$ and the following equation applies:

$$\varphi_{\mathfrak{M}}(v) = \varphi_{\mathfrak{M}}(\bar{x}) + \langle \varphi_{\mathfrak{M}}'(\bar{x}),\ v - \bar{x} \rangle + o(\|v - \bar{x}\|). \tag{5}$$

Let Δ be a vector from space R^{n+s}. We consider a function $\Psi(\Delta)$, the value of which is computed using recursive equations:

$$\begin{aligned} y_i(\Delta) &= \bar{x}_i + \Delta_i, \quad i = 1, \ldots, n, \\ y_k(\Delta) &= f_{\alpha_k}(z_k(\Delta)) + \Delta_k, \quad k = n+1, \ldots, n+s, \end{aligned} \tag{6}$$

where $z_k(\Delta) = (y_{j_1^k}(\Delta), \ldots, y_{j_{l_k}^k}(\Delta))$ and all the other notation has the same meaning as in (3). Moreover $\Psi(\Delta) = y_{n+s}(\Delta)$. We note that $\Psi(0) = \varphi_{\mathfrak{M}}(\bar{x})$.

By the conditions of the theorem, for any k point $z_k(0)$ is an interior point of set $G_{\alpha_k} \cap G'_{\alpha_k}$. Moreover it can be shown that for

sufficiently small Δ the following equation is satisfied: $\|z_k(\Delta)-z_k(0)\|=O(\|\Delta\|)$, $k=1,\ldots,n+s$. From this, in particular, it follows that there exists a neighborhood $B(\bar{x})$ of point $\bar{x}$ such that $\varphi_{\mathfrak{M}}(\cdot)$ is computable at each point $v\in B(\bar{x})$.

We show that for sufficiently small Δ the following formula is true:

$$\Psi(\Delta)=\Psi(0)+\sum_{j=1}^{n+s}p_j\Delta_j+o(\|\Delta\|), \tag{7}$$

where p_j are numbers obtained in (4) when $x=\bar{x}$. And Δ in (7) are said to be sufficiently small if $z_k(\Delta)\in G_{\alpha_k}\cap G'_{\alpha_k}$ for all $k=n+1,\ldots,n+s$. We shall prove equation (7) by induction.

We choose a fairly small vector $\Delta\in R^{n+s}$. We denote by Δ^q the vector with the following components: $\Delta_j^q=0$, $1\leqslant j\leqslant q-1$; $\Delta_j^q=\Delta_j$, $q\leqslant j\leqslant n+s$. We show that for any q, $1\leqslant q\leqslant n+s$:

$$\Psi(\Delta^q)=\Psi(0)+\sum_{j=q}^{n+s}p_j\Delta_j+o(\|\Delta\|). \tag{8}$$

For $q=n+s$ we have $\Psi(\Delta^{n+s})=\Psi(0)+\Delta_{n+s}=\Psi(0)+p_{n+s}\Delta_{n+s}$. We assume that for $q=m+1$ equation (8) is satisfied. We shall prove that it is also fulfilled for $q=m$.

We introduce a vector $\tilde{\Delta}^{m+1}$ with components $\tilde{\Delta}_j^{m+1}=\Delta_j^{m+1}$ for $j\notin\omega_m$, $\tilde{\Delta}_j^{m+1}=\Delta_j+f_{\alpha_j}(z_j(\Delta^m))-f_{\alpha_j}(z_j(\tilde{\Delta}^{m+1}))$ for $j\in\omega_m$ (here ω_m is the set from (4)). It is not hard to see that $y_k(\tilde{\Delta}^{m+1})=y_k(\Delta^m)$ for all $k\neq m$, $\tilde{\Delta}_m^{m+1}=0$. Hence we can assume that (8) is also satisfied for vector $\tilde{\Delta}^{m+1}$ with $q=m+1$. Thus we have:

$$\Psi(\Delta^m) - \Psi(0) = \Psi(\tilde{\Delta}^{m+1}) - \Psi(0) =$$

$$\sum_{j=m+1}^{n+s} p_j \tilde{\Delta}_j^{m+1} + o(\|\Delta\|) = \sum_{\substack{j=m+1 \\ j \notin \omega_m}}^{n+s} p_j \Delta_j + \sum_{j \in \omega_m} p_j \tilde{\Delta}_j^{m+1} + o(\|\Delta\|) =$$

$$\sum_{j=m+1}^{n+s} p_j \Delta_j + \sum_{j \in \omega_m} p_j (f_{\alpha_j}(z_j(\Delta^m)) - f_{\alpha_j}(z_j(\tilde{\Delta}^{m+1}))) + o(\|\Delta\|).$$

We note that in view of condition (1) of theorem 1 for all $j \in \omega_m$:

$$f_{\alpha_j}(z_j(\Delta^m)) - f_{\alpha_j}(z_j(\tilde{\Delta}^{m+1})) = g_j^{i}{}_{mj} \Delta_m + o(\|\Delta\|)$$

(for the definition of $g_j^{i}{}_{mj}$ see (4)). Consequently:

$$\Psi(\Delta^m) - \Psi(0) = \sum_{j=m+1}^{n+s} p_j \Delta_j +$$

$$\sum_{j \in \omega_m} p_j g_j^{i}{}_{mj} \Delta_m + o(\|\Delta\|) = \sum_{j=m}^{n+s} p_j \Delta_j + o(\|\Delta\|).$$

This last equation proves that (8) is true, and therefore (7) is true. It remains to note that (5) is a special case of equation (7) with Δ: $\Delta_j = 0$ for $n+1 \leqslant j \leqslant n+s$. The theorem is thus proved.

Remark. We have shown that, for $k = 1, \ldots, n$, p_k are the partial derivatives of function φ with respect to x_k. In a similar way p_k for $k > n$ can be interpreted as the derivative of φ with respect to y_k, which makes it possible using (7) to evaluate the effect of an absolute error in fulfilling the k-th operation on the absolute error in computing the function.

We note that in view of theorem 1 definition (4) can be viewed as a way of computing the gradient of function $\varphi_{\mathfrak{M}}$. We shall evaluate the difficulty of this method.

We consider two auxiliary functions in two variables: $f_0(x_1, x_2)=x_1+x_2$ and $f_1(x_1, x_2)=x_1x_2$, $m_0=m_1=2$, $G_0=G_1=R^2$. We use the notation $\bar{A}=A\cup\{0\}\cup\{1\}$.

We introduce a function $T(\alpha)$: $\bar{A}\to R^+\equiv\{x\in R^1 \mid x>0\}$, which we shall call the difficulty of operation f_α. By $T(\alpha)$ we shall understand, for example, the time needed to carry out operation f_α on a computer. Below we need the following property of set Ω:

$$\mu\equiv\mu(\Omega)=\sup\left\{\frac{m_\alpha(T(0)+T(1))}{T(\alpha)}\,\middle|\,\alpha\in\bar{A}\right\}.$$

We shall everywhere assume that $\mu<\infty$.

In a similar way we can introduce in set A a function $T_1(\alpha)$: $A\to R^+$, which we shall call the difficulty of simultaneously computing the value and the formal gradient of operation f_α. We assume that:

$$C=\sup\left\{\frac{T_1(\alpha)}{T(\alpha)}\,\middle|\,\alpha\in A\right\}<\infty.$$

Using the functions $T(\alpha)$ and $T_1(\alpha)$ we can define the notion of the difficulty of computing the value of function $\varphi_{\mathfrak{M}}\in\Phi_n$ and the difficulty of computing its formal gradient. In view of (3) the difficulty $T_\varphi(\mathfrak{M})$ of computing the value of function $\varphi_{\mathfrak{M}}$, $\mathfrak{M}=\{\mathfrak{A}, J\}$ is naturally defined as: $T_\varphi(\mathfrak{M})=\sum_{\alpha\in\mathfrak{A}}^{s} T(\alpha)$.

In a similar way in view of (4) the difficulty $T_\varphi'(\mathfrak{M})$ of computing the n-dimensional vector of the formal gradient $\varphi_{\mathfrak{M}}'$ is defined thus:

$$T_\varphi'(\mathfrak{M})=\sum_{\alpha\in\mathfrak{A}}T_1(\alpha)+\sum_{q=1}^{n+s-1}\sum_{t\in\omega_q}(T(0)+T(1)).$$

We note that:

$$\sum_{\alpha\in\mathfrak{A}} T_1(\alpha) \leqslant C \sum_{\alpha\in\mathfrak{A}} T(\alpha) = CT_\varphi(\mathfrak{M}),$$

$$\sum_{q=1}^{n+s-1} \sum_{t\in\omega_q} (T(0)+T(1)) \leqslant \sum_{\alpha\in\mathfrak{A}} m_\alpha (T(0)+$$

$$T(1)) \leqslant \sum_{\alpha\in\mathfrak{A}} \mu T(\alpha) = \mu T_\varphi(\mathfrak{M}).$$

We have thus proved the following theorem.

Theorem 2. If function $\varphi_{\mathfrak{M}}\in\Phi_n$ *is formally differentiable at point* $x\in R^n$, *then the difficulty of computing the n-dimensional vector of the gradient is related to the difficulty of computing this function by inequality:*

$$T_\varphi'(\mathfrak{M}) \leqslant (C+\mu)\, T_\varphi(\mathfrak{M}). \tag{9}$$

We note that constants C and μ in estimation (9) do not depend on the dimension of space n. They are properties of sets Ω and Ω'.

Let set Ω consist of operations $+,-,\times,:$. Then the algorithm for computing gradient of (4) can be written in a more economical form, which eliminates expressions with zeros, multiplications by one, and repeated computations of previously computed magnitudes. The number of operations in this algorithm exceeds the number of operations in the algorithm for computing the function by at most five times.

We now describe a method for computing the second derivatives of function $\varphi_{\mathfrak{M}}\in\Phi_n$. We introduce set Ω'' of second derivatives of the basic operations as follows: with each operation $f_\alpha\in\Omega$ we associate a matrix function:

$$f_\alpha''(x_1,\ldots,x_{m_\alpha}) = \begin{Bmatrix} f_\alpha^{11}(x_1,\ldots,x_{m_\alpha}), \ldots, f_\alpha^{1m_\alpha}(x_1,\ldots,x_{m_\alpha}) \\ \cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots \\ f_\alpha^{m_\alpha 1}(x_1,\ldots,x_{m_\alpha}), \ldots, f_\alpha^{m_\alpha m_\alpha}(x_1,\ldots,x_{m_\alpha}) \end{Bmatrix},$$

where $f_\alpha^{kl} \equiv f_\alpha^{lk}$: $G_\alpha'' \to R$, $k, l = 1, \ldots, m_\alpha$, G_α'' are open sets from R^{m_α}, $\alpha \in A$. We fix an arbitrary point $x \in R^n$ and a direction $u \in R^n$. We consider sequences of numbers $\{y_k\}_{k=1}^{n+s}$, $\{\dot{y}_k\}_{k=1}^{n+s}$, $\{p_k\}_{k=1}^{n+s}$, $\{\dot{p}_k\}_{k=1}^{n+s}$, of vectors $\{g_k\}_{k=n+1}^{n+s}$ $\{g_k\}_{k=n+1}^{n+s}$, and of matrices $\{G_k\}_{k=n+1}^{n+s}$ defined by the recursive equations:

$$
\begin{gathered}
y_i = x_i, \quad i = 1, \ldots, n, \\
y_k = f_{\alpha_k}(z_k), \quad g_k = f'_{\alpha_k}(z_k), \\
G_k = f''_{\alpha_k}(z_k), \quad k = n+1, \ldots, n+s, \\
p_{n+s} = 1, \\
p_q = \sum_{t \in \omega_q} p_t g_t^{iqt}, \quad q = n+s-1, \ldots, 1;
\end{gathered}
\tag{10'}
$$

$$
\begin{gathered}
\dot{y}_i = u_i, \quad i = 1, \ldots, n, \\
y_k = \langle g_k, \dot{z}_k \rangle, \\
g_k = G_k \dot{z}_k, \quad k = n+1, \ldots, n+s, \\
\dot{p}_{n+s} = 0, \\
\dot{p}_q = \sum_{t \in \omega_q} [\dot{p}_t g_t^{iqt} + p_t g_t^{iqt}], \quad q = n+s-1, \ldots, 1,
\end{gathered}
\tag{10''}
$$

where $z_k = (y_{j_1^k}, \ldots, y_{j_{l_k}^k})$, $\dot{z}_k = (\dot{y}_{j_1^k}, \ldots, \dot{y}_{j_{l_k}^k})$ and the rest of the notation is as in (4). We shall say that function $\varphi_{\mathfrak{M}} \in \Phi_n$ is formally twice differentiable at point $x \in R^n$ if in (10') for any k, $n+1 \leqslant k \leqslant n+s$ the following inclusion is satisfied: $z_k \in G_{\alpha_k} \cap G'_{\alpha_k} \cap G''_{\alpha_k}$. Moreover we shall call vector $\varphi_{\mathfrak{M}}''(x, u) = (\dot{p}_1, \ldots, \dot{p}_n)$ the product of the matrix of second formal derivatives on vector u.

We can now prove the following theorem.

Theorem 3. Assume that for any $\alpha \in \mathfrak{A}$ the following conditions are satisfied: (1) function f_α is twice continuously differentiable

in set G_α'': *(2) set* $G_\alpha \cap G_\alpha' \cap G_\alpha''$ *is open; (3) for any* $x \in G_\alpha \cap G_\alpha' \cap G_\alpha''$, *vector* $f_\alpha'(x)$ *is the gradient of function* $f_\alpha(\cdot)$ *while matrix* $f_\alpha''(x)$ *is the matrix of second partial derivatives of function* $f_\alpha(\cdot)$ *at point x. Then if function* $\varphi_{\mathfrak{M}}(\cdot)$ *from* Φ_n *is formally twice differentiable at point* $\bar{x}$, *then it is twice differentiable at point* $\bar{x}$ *and vector* $(\dot{p}_1, \ldots, \dot{p}_n)$ *is equal to the product of the matrix of second derivatives on vector u.*

The proof of theorem 3 is similar to that of theorem 1.

Corollary 1. If for any $\alpha \in \mathfrak{A}$ function f_α is defined and is twice continuously differentiable in the whole space R^{m_α} and for any $x \in R^{m_\alpha}$, $f_\alpha'(x)$, and $f_\alpha''(x)$ are respectively the gradient and matrix of second derivatives of function $\varphi_\alpha(\cdot)$ at point x, then function $\varphi_{\mathfrak{M}}(\cdot)$ is twice continuously differentiable in the whole space R^n and $\varphi_{\mathfrak{M}}''(\bar{x}, u)$ is the product of the matrix of the second partial derivatives of function $\varphi_{\mathfrak{M}}(\cdot)$ at point $\bar{x} \in R^n$ in direction $u \in R^n$.

We evaluate the difficulty of computing vector $\varphi''(x, u)$ using formulae (10′) and (10′′). For this purpose we introduce a function $T_2(\alpha)$: $A \to R^+$ which we call the difficulty of simultaneously computing the value, the formal gradient, and the matrix of second formal derivatives of operation f_α.

We use notation: $C' = \sup\{T_2(\alpha)/T(\alpha) \mid \alpha \in A\}$. Moreover, let T_α' be the difficulty of simultaneously computing the value of the quadratic function and its gradient in space R^{m_α} and let $\mu' = \sup\{T_\alpha'/T(\alpha) \mid \alpha \in A\}$. Then $T_\varphi''(\mathfrak{M})$—the difficulty of computing vector $\varphi_{\mathfrak{M}}''(x, u)$—can be evaluated as:

$$T_\varphi''(\mathfrak{M}) = \sum_{\alpha \in \mathfrak{A}} T_2(\alpha) + \sum_{\alpha \in \mathfrak{A}} T_\alpha' + 3 \sum_{q=1}^{n+s-1} \sum_{t \in \omega_q} (T(0) + T(1)) \leqslant$$

$$(C' + \mu') \sum_{\alpha \in \mathfrak{A}} T(\alpha) + 3 \sum_{q=1}^{n+s-1} m_{\alpha_q} (T(0) + T(1)) \leqslant$$

$$(C' + \mu' + 3\mu)\, T_\varphi(\mathfrak{M}).$$

We note that in computing the product of the matrix of second formal derivatives of function $\varphi_{\mathfrak{M}}(x)$ and the different vectors

there is no need to repeat the calculations using formulae (10′). It is therefore possible to organize the computation of the matrix of second formal derivatives as follows: carry out the calculations using formulae (10′) once, and then use formulae (10′′) n times, choosing $u=e_k$, where e_k is the k-th coordinate vector in space R^n. The difficulty $\overline{T}''(\mathfrak{M})$ of computing the matrix of second formal derivatives of function $\varphi_{\mathfrak{M}}(x)$ by this algorithm is estimated as:

$$\overline{T}_{\varphi}''(\mathfrak{M}) = \sum_{\alpha\in\mathfrak{A}} T_2(\alpha) + \sum_{\alpha\in\mathfrak{A}} T_\alpha' +$$

$$\sum_{q=1}^{n+s-1} \sum_{t\in\omega_q} (T(0)+T(1)) + n\Big[\sum_{\alpha\in\mathfrak{A}} m_\alpha(m_\alpha+1)(T(0)+$$

$$T(1)) + 2\sum_{q=1}^{n+s-1} \sum_{t\in\omega_q} (T(0)+T(1))\Big] \leqslant (C'+\mu'+\mu)\,T_\varphi(\mathfrak{M}) +$$

$$n(T(0)+T(1))\sum_{\alpha\in\mathfrak{A}} m_\alpha(m_\alpha+3).$$

An analysis of formulae (10′) and (10′′) enables us to draw the following important conclusion: all information concerning the matrix of second formal derivatives of function $\varphi_{\mathfrak{M}}(x)$ is stored in "stripped-down" form in sets $\{y_k\}$, $\{g_k\}$, $\{G_k\}$, and $\{p_k\}$. The difficulty of getting this information is a magnitude of order $O(T_\varphi(\mathfrak{M}))$. The dimension of space n only comes into the estimation of the number of additions and multiplications necessary to "collect" this information into an $n\times n$ matrix.

We now combine these estimations of the difficulty of computing the second derivatives in the following theorem.

Theorem 4. If function $\varphi_{\mathfrak{M}}$ from Φ_n is formally twice differentiable at point $x\in R^n$ then:

(1) the following estimation applies for the difficulty of computing the product of the matrix of second-order formal derivatives of $\varphi_{\mathfrak{M}}$ at point x and direction $u\in R^n$:

$$T_\varphi''(\mathfrak{M}) \leqslant (C'+\mu'+3\mu)\, T_\varphi(\mathfrak{M}); \tag{11}$$

(2) the following estimation applies for the difficulty of computing the matrix of second formal derivatives of function $\varphi_{\mathfrak{M}}$ *at point* x*:*

$$\bar{T}_\varphi''(\mathfrak{M}) \leqslant (C' + \mu' + \mu)\, T_\varphi(\mathfrak{M}) + n\,(T(0) + T(1)) \sum_{\alpha \in \mathfrak{A}} m_\alpha (m_\alpha + 3). \tag{12}$$

We note that the dimensions m_α of operations f_α of set Ω do not depend in any way on the dimension of the functions from set Φ_n. In fact if we take as the set of basic operations Ω all the arithmetic operations and operations corresponding to the elementary functions, then Φ_n is the set of functions of n variables written in the form of a finite sequence of superpositions of operations from Ω. Here the maximum value of m_α for the operations from this set is equal to two. In this case we can rewrite estimation (12) as:

$$\bar{T}_\varphi''(\mathfrak{M}) \leqslant (C'+\mu'+\mu)\, T_\varphi(\mathfrak{M}) +$$

$$10ns\,(T(0)+T(1)) = O(nT_\varphi(\mathfrak{M})).$$

We now discuss the method with which we obtained our algorithm for computing second formal derivatives of function $\varphi_{\mathfrak{M}}$.

We consider the vector function $x(\beta) = \bar{x} + \beta u : R \to R^n$, $\bar{x}, u \in R^n$. Assume that we have to compute the derivative of vector function $f'(x(\beta))$ with respect to β, where $f : R^n \to R$ is a twice differentiable function. It is not hard to see that $(f'(x(\beta)))_\beta' = f''(x(\beta))\,u$. It is intuitively clear that, if there is a sequence of operations with which vector $f'(x)$ is computed, then by formally differentiating this sequence of operations (program) with respect to β we derive a program for computing vector

$f''(x(\beta))u$. We note only that if in the program for computing $f'(x)$ there were only one input (vector x), then in the program for computing $f''(x)u$ there will be two (vectors x and $\dot{x}=u$). This method was also used to derive algorithm (10′) and (10′′) from algorithm (4). It would be possible to proceed in exactly the same way in constructing algorithms for computing derivatives of higher orders. This possibility is not discussed in detail in the present article.

We now discuss the reason which made us use the term "formal differentiation" in all the definitions and theorems of this section. The point is that all the algorithms considered above are algorithms for differentiating programs of some special type. In fact set $\mathfrak{M}=\mathfrak{M}(s)=\{\mathfrak{A}, J\}$, which specifies a rule for computing the value of function $\varphi_{\mathfrak{M}}$, is a program in which there are s instructions, such that in the k-th command α_k, $\alpha_k \in \mathfrak{A}$ is the operation code, I_k, $I_k \in J$ is the list of addresses of operands, and y_k is the working cell into which the result of the operations is transferred.

The rules for formal differentiation can be applied to any correct program (or set) $\mathfrak{M}$ where the values of all the operands of any operation f_{α_k} are computed before fulfilling this operation. Here if all f_{α_k}, $\alpha_k \in \mathfrak{A}$ are differentiable and f_{α_k} are their gradients, then we derive a program for computing the gradient of function $\varphi_{\mathfrak{M}}$. But the case may arise where some command $\bar{\alpha}$ from $\mathfrak{A}$ is not differentiable. It turns out in this case that it is possible to define the formal gradient of operation $f_{\bar{\alpha}}$ in such a way that the formal gradient of function $\varphi_{\mathfrak{M}}$ will have a sensible meaning: for example, if $f_{\bar{\alpha}}(x) = |x|$, $x \in R$, set $f_{\bar{\alpha}}'(x) = \operatorname{sign} x$. The issues arising in this connection are not discussed in detail here. We limit ourselves to stating the possibility of using the algorithms proposed to compute certain different analogs of derivatives for nondifferentiable functions.

To conclude this section we note that a method has actually been proposed which enables us to use a program for computing the value of function $\mathfrak{M}$ to establish a program for computing the derivatives of this function. The proposed method can easily be automated. When the derivatives are computed by automated

methods (or in any other way) some complications may arise from the fact that at particular points the derivative actually exists, even though in carrying out the computations formally it is necessary to identify indeterminacy of the type $0/0$, $0\cdot\infty$, and so on. An example of this is provided by the function $f(x, y)=\sqrt[4]{x^4+y^4}$ at point $(0, 0)$. However, these problems are eliminated using purely programming methods.

3. Using automated differentiation algorithms to solve extremal problems

Section 2 established some estimations of the difficulty of computing derivatives for a broad class of non-linear functions. It is obvious that the estimations derived cannot be substantially improved in terms of their difficulty. At the same time Section 2 proposed algorithms which realize these estimations. Of course, it is possible in the case of a particular non-linear function to write a program for computing its derivatives which exhibits the same difficulty as the algorithms we have described. But in the case of a fairly cumbersome and complicated function writing such a program is a complex problem even for an experienced programmer. For this reason the algorithms of Section 2 yield the greatest return if they are implemented in automated form (let us say, after a program for computing the function has been compiled by the operator). We note that in the Central Mathematical Economics Institute (TsEMI) a system—DIANA—has been developed for dialog analysis of non-linear functions, which implements the algorithms we have outlined for computing the derivatives. The system has been implemented on a NORD-100 computer.

Apart from the automation effect, which substantially lightens the task of data preparation and reduces the actual computation time for particular problems on the computer, the algorithms in Section 2 enable us to change our quantitative views concerning

the difficulty of different methods for solving extremal problems. We consider some of them.

1. Cubic interpolation in one-dimensional search methods

In solving auxiliary problems of one-dimensional minimization in non-linear programming, [we find that] one of the most efficient techniques is the cubic interpolation procedure (see [2]). but for it to work it is necessary at each stage to compute the value of the objective function and its gradients. Here the gradient is used only to compute the directional derivative. In connection with this "irrational" use of the gradient, a number of authors, relying on relation (1), have argued against this procedure. Using the algorithm for automated computation of the gradient (4) removes such objections, as is clear from (9).

2. The method of conjugate gradients

The various frameworks of conjugate gradient methods intended to solve unconstrained problems of minimizing a non-linear function $f(x)$, $x \in R^n$, include the following (see [3]):

$$\begin{gathered} x_{k+1}=x_k-\beta_k s_k, \qquad k=0, 1, \ldots, \\ s_0=f'(x_0), \qquad s_k=f'(x_k)-\xi_k s_{k-1}, \qquad k=1, 2, \ldots, \\ \xi_k=\frac{\langle f''(x_k)s_{k-1}, f'(x_k)\rangle}{\langle f''(x_k)s_{k-1}, s_{k-1}\rangle} \\ \beta_k=\arg\min\{f(x_k-\beta s_k)\mid \beta\geqslant 0\}, \qquad k=0, 1, \ldots. \end{gathered} \tag{13}$$

However, framework (13) has been little studied and used in practice because of the need to implement the formula for computing the ξ_k (again the impact of plausible estimations (1)!). But if we use automated differentiation algorithm (10′)–(10″) to compute these values, then in view of estimation (11) the difficulty of computing ξ_k will be of the order of difficulty of computing the value of the objective function. Hence in this case framework

(13) in terms of its *a priori* indicators turns out to be fully competitive with other frameworks for the conjugate gradients method.

3. The Newton method

Let function $f(x)$, $x \in R^n$ be twice continuously differentiable and concave in the whole of R^n. If we adopt the Newton method to solve the problem of minimizing $f(x)$ on R^n, then at each stage we have to solve the system of linear equations:

$$f''(x)\,s = -f'(x). \tag{14}$$

We estimate the overall difficulty of computing matrix $G=f''(x)$ and vector $g=f'(x)$ and of solving system (14) in the case where it is possible to use differentiation algorithm (10′)–(10′′).

Solving (14) comes down to minimizing quadratic function:

$$Q(s) = \frac{1}{2}\langle Gs, s\rangle + \langle g, s\rangle. \tag{15}$$

We use the conjugate gradient method to solve problem (15). In order to obtain a precise solution $s^* = -G^{-1}g$, this method requires n iterations. At each of them it will be necessary to compute the gradient of function $Q(s)$ $(Q'(s) = Gs+g = f''(x)\,s+f'(x))$ and the scalar product $\langle Gu, u\rangle = \langle f''(x)\,u, u\rangle$ (in order to find a one-dimensional minimum) and to recompute two n-dimensional vectors.

Thus in view of estimations (11), (12) the overall difficulty of computing the Newtonian direction does not exceed:

$$(C' + \mu' + \mu)\,T_\varphi(\mathfrak{M}) + 2n\,(T(0) +$$

$$T(1))\sum_{\alpha \in \mathfrak{A}} m_\alpha\,(m_\alpha + 3) + O(n^2) = O(nT_\varphi(\mathfrak{M}));$$

i.e., the difficulty of computing a Newtonian step using method (10′) and (10′′) turns out to be of an order equal to that of once computing a difference approximation of the gradient.

4. Optimal control problems

The proposed algorithm can be used to solve an optimal control problem. It is interesting to note that, for some problems of a special type, it is identical with known methods of computing the gradient (for example, computing the gradient of a function using a conjugate system [4]). At the same time for other problems this method turns out to be better than the traditional one.

Let us consider the following example. Assume that in problem:

$$\min \{f(a) \mid a \in R^n\}$$

function $f(a)$ is computed using recursive equations:

$$f(a) = \varphi(x_N),$$
$$x_{k+1} = \varphi_k(x_k, a), \quad k = 0, \ldots, N-1,$$

where $x_k \in R^m$ and x_0 is fixed. Then the difficulty of the natural method of computing gradient:

$$f'(a) = V_N \varphi_x'(x_N),$$
$$V_{k+1} = V_k (\varphi_k)_x'(x_k, a) + (\varphi_k)_a'(x_k, a), \quad k = 0, \ldots, N-1,$$

(where V_k, $(\varphi_k)_x'$, and $(\varphi_k)_a'$ are matrices of dimensions $n \times m$, $m \times m$, and $n \times m$ respectively) depends on the difficulty of computing the product of these matrices.

At the same time the present article implies a method of computing $f'(a)$, the difficulty of which depends only on the difficulty of computing function φ_k.

References

1. F. W. Pfeiffer, "Some Advances Related to NLP," *SIGMAP Bulletin*, 1980, no. 28.
2. D. Himmelbray, *Applied Linear Programming* (Russian edition), Moscow, "Mir" Publishers, 1975.
3. J. W. Daniel, *The Approximate Minimization of Functionals*, New York, 1971.
4. Iu. G. Evtushenko, *Metody resheniia ekstremal'nykh zadach i ikh primenenie v sistemakh optimizatsii*, Moscow, "Nauka" Publishers, 1982.

Received March 2, 1983