Journal of Computational Mathematics, Vol.10, No.2, 1992, 129–146.

NEW ODE METHODS FOR EQUALITY CONSTRAINED OPTIMIZATION (2) — Algorithms*

Pan Ping-qi
(*Nanjing Forestry University, Nanjing, China*)

Abstract

As a continuation of [1], this paper considers implementation of ODE approaches. A modified Hamming's algorithm for integration of (ECP)-equation is suggested to obtain a local solution. In addition to the main algorithm, three supporting algorithms are also described: two are for evaluation of the right-hand side of (ECP)-equation, which may be especially suitable for certain kinds of (ECP)-equation when applied to large scale problems; the third one, with a convergence theorem, is for computing an initial feasible point. Our numerical results obtained by executing these algorithms on an example of (ECP)-equation given in [1] on five test problems indicate their remarkable superiority of performance to Tanabe's ODE version that is recently claimed to be much better than some well-known SQP techniques.

This work is a continuation of [1], so the same notation is used as before and the section numbers are continued. Implementation problems of ODE methods are considered in detail here. In Section 4, The main algorithm, a modified Hamming's algorithm for integration of (ECP)-equation, is described . Two supporting algorithms for evaluation of the right-hand side of (ECP)-equation are presented in Section 5, which may be especially suitable for certain kinds of (ECP)-equation when applied to large scale problems. Then, in Section 6, an algorithm for computing an initial feasible point and a related convergence theorem are given. Finally, Section 7 presents our numerical results obtained by executing these algorithms on an example of (ECP)-equation proposed in [1] on five test problems. These results show their remarkable superiority of performance to Tanabe's ODE version that is recently claimed to be much better than some well-known SQP techniques[2].

§4. Numerical Integration of (ECP)-Equation

Without loss of generality, from now on we will consider the implementation problems of ODE approaches only for minimization problem (ECP). Once an (ECP)-equation has been chosen and an initial point $x_0 \in \hat{X}$ has been obtained, what we

* Received October 27, 1988..

need to do next is just to handle it numerically. Let the ECP-equation be of the form:

$$\begin{cases} \dfrac{dx}{dt} = -\varphi(x)P(x)A(x)\nabla f(x) \equiv p(x), & (4.1a) \\ x(0) = x_0. & (4.1b) \end{cases}$$

In the sequel, we will always assume that (ECP)-equation (4.1) has a complete-limit point x^*, which we refer to as limit point for short. Let us choose the (ECP)-rate factor (3.13), i.e., $\varphi(x) = 1/\|PA\nabla f\|_2$ so that the right-hand side is standard-ized:

$$\|p(x)\|_2 = 1. \tag{4.2}$$

As mentioned at the end of Section 3, this gives the easiest may to get to know the length of the trajectory from x_0 to any point $x(t)$ along the trajectory.

For (4.1) we can choose between a number of numerical integration schemes with differing techniques of step-size control. While it is impossible at this stage to specify a best integration scheme for our problem, it may be proper to use a higher order integration method here than in the unconstrained case, where approaches of Euler's type are often suggested, for now the search should stay closer to the trajectory (see Brown and Bartholomew [2]). Furthermore, since what we are really interested in is the limit point, rather than the trajectory itself, the integration method should possess a good round-off property, and its computation complexity should at the same time be as low as possible. In the light of these, Hamming's approach[4], a fourth order, multistep and predictor-corrector method, seems to be among the reasonable choices. We will incorporate it into a stepsize-variable integration algorithm.

Denote simply the k-th stepsize, $x(t_k)$ and $p(x(t_k))$ by α_k, x_k and p_k, respectively. Suppose that other three points x_1, x_2 and x_3 are ready to be used besides x_0. Then at the k-th step, $k = 3, 4, \cdots$, point x_{k+1} may be calculated by applying Hamming's formulas:

$$\left.\begin{aligned} &\text{(1) Prediction } u_{k+1} = x_{k-3} + \frac{8}{3}\alpha_k(p_k + p_{k-2} - \frac{1}{2}p_{k-1}) \\ &\qquad v_{k+1} = u_{k+1} - \frac{112}{121}d_k \ (d_3 = 0) \\ &\qquad w_{k+1} = p(v_{k+1}) \\ &\text{(2) Correction } c_{k+1} = \frac{1}{8}[9x_k - x_{k-2} + 3\alpha_k(w_{k+1} + 2p_k - p_{k-1})] \\ &\qquad d_{k+1} = u_{k+1} - c_{k+1} \\ &\qquad x_{k+1} = c_{k+1} + \frac{9}{121}d_{k+1} \\ &\qquad p_{k+1} = p(x_{k+1}) \end{aligned}\right\} \tag{4.3}$$

This method visits the right-hand side of the equation twice per step. In addition, it is advantageous in that the local truncation error of the method can be simply controlled through $\|d_{k+1}\|_\infty$.

The following is to ensure sufficient accuracy of the integration:

Criterion 4.1 (Acceptance of steps). If (4.4) below is satisfied, accept point x_{k+1} and proceed to the next step:

$$\begin{cases} f(x_{k+1}) \le f(x_k), & \text{(4.4a)} \\ \|d_{k+1}\|_\infty < \varepsilon_1, & \text{(4.4b)} \end{cases}$$

where ε_1 is a given positive number.

In the case when (4.4) is not satisfied, what we will do is to cut down by half the stepsize to improve the precision of the integration. We will take two additional values

$$t_a = (t_{k-2} + t_{k-1})/2 \quad \text{and} \quad t_b = (t_{k-1} + t_k)/2$$

of t (see Fig. 1), and compute corresponding points x_a and x_b on the trajectory and p_b through the following formulas:

$$x_a = (x_{k-2} + x_{k-1})/2 + \alpha_k(p_{k-1} - p_k)/8, \quad \text{(4.5a)}$$

$$x_b = (x_{k-1} + x_k)/2 + \alpha_k(p_{k-1} - p_k)/8, \quad \text{(4.5b)}$$

$$p_b = 3(x_k - x_{k-1})/2(\alpha_k) - (p_{k-1} + p_k)/4 \quad \text{(4.5c)}$$

which are derived by using the Hermite interpolation formula (see, for example, Powell [11]). We set new $x_{k-3}, x_{k-2}, x_{k-1}, p_{k-2}$ and p_{k-1} to $x_a, x_{k-1}, x_b, p_{k-1}$ and p_b, respectively, and then set

$$\alpha_k = \alpha_k/2$$

and compute x_{k+1} again. This procedure is repeated until (4.4) is fulfilled.

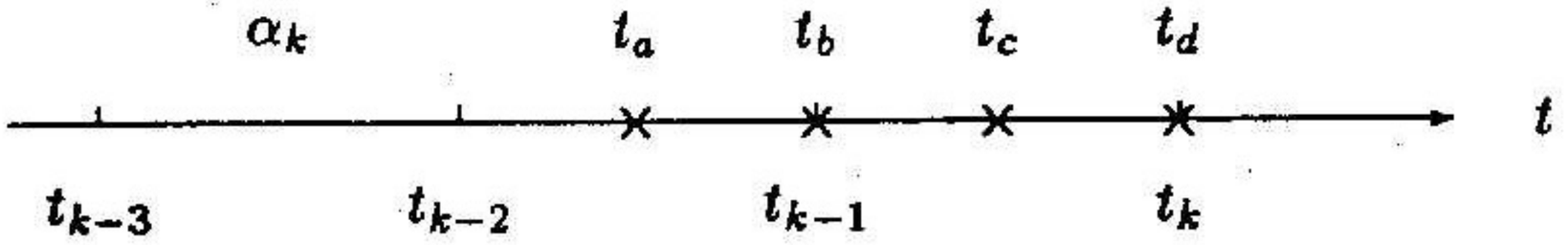

Fig. 1. Step size change

It may happen practically, however, that at some k-th step, no matter how many times the procedure is repeated and how small the stepsize is, as a result of the effect of rounding errors, condition (4.4) is not satisfied. No further progress is

then possible and the integration must be abandoned. This situation is unlikely to occur unless x_k is already close to x^*. In practice the procedure is repeated several times (between 5 and 10) if necessary, and after that x_k is accepted as the final estimate of x^*. But in general, it is not necessary to have such an accurate one; so a practical convergence criterion is needed. We will return to this later.

The above numerical integration procedures may be put in the algorithm below:

Algorithm 4.1 (Integration of ECP-equation (4.1)). Given $x_0 \in \hat{X}, \alpha_0 > 0$ and $\varepsilon_1 > 0$.

(1) Set $k = 2, d = 0$ and $j = 0$, and compute $f = f(x_0)$.

(2) Apply Runge-Kuttas' method on the ECP-equation (4.1), compute x_1, x_2 and x_3, and corresponding $p_1 = p(x_1), p_2 = p(x_2)$ and $p_3 = p(x_3)$.

(3) Compute $f_1 = f(x_{k+1})$; if $f_1 < f$, go to (9).

(4) If $k = 2$, set $\alpha_0 = \alpha_0/2$, and go to (2).

(5) If $j > 5$, set $x^* = x_k$, and go to (14).

(6) Set $j = j + 1$.

(7) Compute (4.5), and set $x_{k-3} = x_2$, $x_{k-2} = x_{k-1}$, $x_{k-1} = x_b$, $p_{k-2} = p_{k-1}$ and $p_{k-1} = p_b$ in succession.

(8) Set $d_k = 0$, $\alpha_k = \alpha_k/2$, and go to (11).

(9) If a practical convergence is attained, go to (13).

(10) Set $k = k + 1$, $f = f_1$, $j = 0$ and $\alpha_k = \alpha_{k-1}$.

(11) Compute (4.3).

(12) If $\|d_{k+1}\|_\infty < \varepsilon_1$, go to (3); otherwise go to (7).

(13) Set $x^* = x_{k+1}$.

(14) Stop.

Note. It would be proper in practice to include in Algorithm 4.1 a device of restarting to handle the case when the accuracy of a solution approximation is found not to be within a specified tolerance.

The following is supplied for step (9) of Algorithm 4.1:

Criterion 4.2 (Practical convergence). Terminate Algorithm 4.1 if the stepsize becomes sufficiently small, satisfying

$$\alpha_k < \varepsilon \tag{4.6}$$

where $\varepsilon \ll 1$ is a given positive number.

The simplicity of Criterion 4.2 results from (4.2) (by using the ECP-rate factor (3.13)). This can be explained as follows. Suppose that (4.6) is satisfied at some k-th step of Algorithm 4.1. Noting $x_{k+1} = x(t_{k+1})$ and $x_k = x(t_k)$, we have by Remark 3.5 that

$$\|x_{k+1} - x_k\|_2 \approx \alpha_k < \varepsilon. \tag{4.7}$$

Since ε is sufficiently samll, it is not possible to make significant progress, and it is likely to be near x^* already. Our experience is that the performance of Criterion 4.2 is very satisfactory in practice.

Even the criterion is not satisfied, it is often necessary to terminate the integration if the number NR of steps becomes larger than a given positive integer N, i.e.,

$$NR > N, \tag{4.8}$$

partly because sometimes the trajectory of (4.1) may be unbounded, or may have an infinite length.

§5. Evaluation of the Right-Hand Side

In this section some techniques for the evaluation of $p(x)$ in (4.1) are considered. We assume that explicit expressions for $f(x), h(x)$ and their first order derivatives are supplied. It is often the case, for example when one of the ECP-direction matrices given in Section 3 is used, that the core of the evaluation of $p(x)$ comprises the two products

$$\text{(i) } (\nabla^2 f(x))b, \quad \text{(ii) } P(x)b, \tag{5.1}$$

where $\nabla^2 f(x)$ and $P(x)$ are the Hessian of $f(x)$ and the projection matrix determined by (2.1) evaluated at some x, respectively; and $b \in R^n$ is a given vector.

For the computation of product (i), the simple approximate formula below can be employed:

$$(\nabla^2 f(x))b \approx [\nabla f(x + \hat{\sigma} b) - \nabla f(x)]/\hat{\sigma}, \tag{5.2}$$

where $\hat{\sigma} = \sigma/\|b\|_2$ and σ is a sufficiently small positive number. The calculation of $\nabla^2 f(x)$ is then bypassed in this way.

It is much more complicated to deal with product (ii) because of the involvement of the inverse or the generalized inverse in (2.1). By Q–R decomposition, Theorem 5.1 below will help make a detour. In the following, the matrices and vectors are all evaluated at some point x.

Theorem 5.1. *Let P be the projection matrix given by* (2.1), *J be the transpose of the Jacobian of h and $b \in R^n$ be an arbitrary vector. Suppose that an orthogonal matrix $Q \in R^{n\times n}$ has been computed such that*

$$QJ = \hat{J} = \begin{bmatrix} R \\ \cdots \\ 0 \end{bmatrix}_{n-m}^{m} \tag{5.3}$$

zeros below the diagonal elements in the order of successive columns, and finally we have

$$(D_{mn}S_{mn}\,D_{m\,n-1}^{-1})(D_{m\,n-1}S_{m\,n-1}D_{m\,n-2}^{-1})\cdots(D_{13}S_{13}D_{12}^{-1})(D_{12}S_{12})J$$
$$=(D_{mn}S_{mn}\cdots S_{12})J \tag{5.16}$$

being upper triangular. It is also evident that

$$Q=D_{mn}S_{mn}\cdots S_{12} \tag{5.17}$$

is orthogonal. In order to improve numerical stability and to avoid overflow, the S_{ij} and D_{ij} in the above procedures are constructed via Theorem 5.2 or 5.3 according as $d_iz_{ii}^2\geq d_jz_{ji}^2$ or not. In practice, instead of explicitly forming Q, it is economical to leave it in factored form and to access it through S_{ij} and D_{mn}. In addition, the corresponding S_{ij} is not needed whenever $z_{ji}=0$; hence the number ns of transformations is no greater than the number of elements below the diagonal of J, i.e.,

$$ns=m(m-1)/2+m(n-m)=m(2n-m-1)/2. \tag{5.18}$$

Taking into account all of the above considerations, we obtain Algorithm 5.1. Two working vectors $\tilde{a}=(\alpha_k)\in R^{ns}$ and $\tilde{b}=(\beta_k)\in R^{ns}$ are introduced in it to store α and β at every k-th transformation, and another vector $d=(d_k)\in R^n$ to store the diagonal elements of D_{ij}. At each transformation, once z_{ji} is transitted to zero, its location becomes free and is then set to 1 or 2, to record which of Theorems 5.2 and 5.3 has been used.

Algorithm 5.1. (Fast Givens Orthogonalization). Given $J=(z_{ij})\in R^{n\times m}$.

(1) Set $i=0,k=0$ and $d=1$ for $l=1,\cdots,n$.

(2) Set $i=i+1$, and then $j=i$.

(3) Set $j=j+1$.

(4) If $z_{ji}=0$, go to (8).

(5) Set $k=k+1$.

(6) If $d_iz_{ii}^2>d_jz_{ji}^2$, then compute (5.8), (5.10) and (5.11), and set $z_{ji}=1$; else compute (5.12), (5.14) and (5.15), and set $z_{ji}=2$.

(7) Set $d_i=\hat{d}_i,d_j=\hat{d}_j,\alpha_k=\alpha,\beta_k=\beta,z_{il}=\hat{z}_{il}$ for $l=i,\cdots,m$, and $z_{jl}=\hat{z}_{jl}$, for $l=i+1,\cdots,m$.

(8) If $j<n$, go to (3).

(9) If $i<m$, go to (2).

(10) Stop.

Noting (5.4), (5.5) and (5.17), we obtain Pb in terms of S_{ij} and D_{mn}:

$$Pb = S_{12}^T S_{13}^T \cdots S_{m\,n-1}^T S_{mn} \begin{bmatrix} 0 \\ \cdots \\ u_2 \end{bmatrix}, \tag{5.19}$$

where u_2 is determined by

$$\begin{bmatrix} 0 \\ \cdots \\ u_2 \end{bmatrix} = D_{mn} S_{mn} S_{m\,n-1} \cdots S_{12} b. \tag{5.20}$$

Algorithm 5.2 below gives Pb via (5.19) along with (5.20). In the algorithm the output of Algorithm 5.1, i.e. $D = (d_i) \in R^n$, $J = (z_{ij}) \in R^{n\times m}$, $\tilde{a} = (\alpha_k) \in R^{ns}$ and $\tilde{b} = (\beta_k) \in R^{ns}$ are supplied as the input; and finally, Pb overwrites b.

Algorithm 5.2 (Computation of Pb). Given $b = (b_i) \in R^n$.

(1) Set $i = 0$, and $k = 0$.

(2) Set $i = i + 1$, and then $j = i$.

(3) Set $j \leftarrow j + 1$.

(4) If $z_{ji} = 0$, go to (8).

(5) Set $k = k + 1$.

(6) If $z_{ji} = 1$, then compute $q = b_i + \alpha_k b_j, b_j = \beta_k b_i + b_j$; else compute $q = \alpha_k b_i + b_j$, $b_j = b_i + \beta_k b_j$.

(7) Set $b_i = q$.

(8) If $j < n$, go to (3).

(9) If $i < m$, go to (2).

(10) Set $b_i = 0$, for $i = 1, \cdots, m$, and $b_j = b_j d_j$, for $j = m + 1, \cdots, n$.

(11) If $z_{ji} = 0$, go to (14).

(12) If $z_{ji} = 1$, then compute $q = b_i + \beta_k b_j$ and $b_j = \alpha_k b_i + b_j$; else compute $q = \alpha_k b_i + b_j$ and $b_j = b_i + \beta_k b_j$.

(13) Set $b_i = q$ and $k = k - 1$.

(14) Set $j = j - 1$; if $j > i$, go to (11).

(15) Set $i = i - 1$ and $j = n$; if $i > 0$, go to (11).

(16) Stop.

§6. Determination of the Initial Point

What remains to be done now is to gain a good estimate of an initial point. A new approach for doing this is presented in this section.

Theorem 6.1. *Let $h(x) : R^n \to R^m$ and $\hat{x}_0 \in R^n$ be given and suppose that $h(\hat{x}_0) \neq 0$, $g(x) = [h(x)]^T h(x)$, and $h(x)$ is continuously differentiable on the convex hull of the level set*

$$S(g, \hat{x}_0) = \{x |\ x \in R^n, g(x) \leq g(\hat{x}_0)\}. \tag{6.1}$$

If for all $x \in S(g, \hat{x}_0)$ such that $h(x) \neq 0$ the transpose $J(x)$ of the Jacobian of $h(x)$ has full column rank m, $[r_i(x)]^T$ is the i-th row vector of $J(x)$ and

$$D(x) = \operatorname{diag}(d_1(x), \cdots, d_n(x)), \tag{6.2}$$

where

$$d_i(x) = \begin{cases} 1/\| r_i(x) \|_\infty, & \text{if } r_i(x) \neq 0, \\ 1, & \text{otherwise,} \end{cases} \tag{6.3}$$

then the vector

$$q(x) = -(1/\| h(x) \|_\infty) D(x) J(x) h(x) \tag{6.4}$$

is a descent direction for g at x.

Proof. In the following, the matrices and vectors are all evaluated at x. Note that the gradient of g is

$$\nabla g = 2Jh, \tag{6.5}$$

$h \neq 0$ and J has full column rank m. The matrix D is clearly positive definite. Hence we obtain from (6.4) and (6.5) that

$$(\nabla g)^T q = -(2/\|h\|_\infty)(Jh)^T D(Jh) < 0. \tag{6.6}$$

Theorem 6.1 is thus proved.

Let $\hat{x}_k \in S(g, \hat{x}_0)$ be the k-th approximation of an initial point and $h(\hat{x}_k) \neq 0$. Clearly, by Theorem 6.1,

$$q(\hat{x}_k) = -[D(\hat{x}_k) J(\hat{x}_k)](h(\hat{x}_k)/\|h(\hat{x}_k)\|_\infty) \tag{6.7}$$

is a descent direction for g at $\hat{x}_k$. Notice that for numerical purpose both the matrix in the bracket [ ] and the vector in the bracket ( ) are standardized in some sense, while (6.7) is an expression equivalent to (6.4). Our approach for computation of an initial point x_0 is contained in the following:

Algorithm 6.1 (Computation of x_0). Given an estimate $\hat{x}_0$ of x_0 and $\varepsilon_0 > 0$.
(1) Set $k = 0$.
(2) Compute $g_k = g(\hat{x}_k)$ from (6.1); if $g_k < \varepsilon_0$, go to (7).
(3) Compute $q_k = q(\hat{x}_k)$ from (6.7), (6.2) and (6.3).

(4) Comput the smallest $\beta_k > 0$ such that

$$g(\hat{x}_k + \beta_k q_k) = \min_{\beta>0}\{g(\hat{x}_k + \beta q_k)\}. \tag{6.8}$$

(5) Compute $\hat{x}_{k+1} = \hat{x}_k + \beta_k q_k$.
(6) Set $k = k + 1$, and go to (2).
(7) Set $x_0 = \hat{x}_k$.
(8) Stop.

The global convergence of the preceding algorithm is given below, while its convergence rate will not be considered here.

Theorem 6.2. *Under the same assumptions as in Theorem* 6.1, *if, in addition, the level set* $S(g, \hat{x}_0)$ *is bounded, then Algorithm* 6.1 *with* $\varepsilon_0 = 0$ *generates a sequence* $\{\hat{x}_k\}$ *that is either finite, terminating at some* $\hat{x}_k$ *such that* $h(\hat{x}_k) = 0$, *or infinite, having at least one cluster point. If, in addition, at some cluster point, say* $\hat{x}$, *each row vector* $[r_i(\hat{x})]^T$ *of* $J(\hat{x})$ *is nonzero, then* $h(\hat{x}) = 0$.

Proof. It is clear under the assumptions that $g(x)$ is continuously differentiable on the convex hull of $S(g, \hat{x}_0)$, and $h(x) = 0$ if and only if $\nabla g(x) = 0$ for all $x \in S(g, \hat{x}_0)$. Note also that $D(x)/(2\|h(x)\|_\infty)$ is positive definite at $x \in S(g, \hat{x}_0)$ such that $h(x) \neq 0$ and that each row vector of $J(\hat{x})$ being nonzero implies that $D(\hat{x})$ is continuous at $\hat{x}$. Then, the remainder of the proof is an analogy to that of [9] for an algorithm in which search direction $-B(x)\nabla f(x)$ is continuous and $B(x)$ is positive definite, though f should be set to g and $B(x)$ to $D(x)/(2\|h(x)\|_\infty)$ in our case.

Remark 6.1. Theorem 6.2 is also valid for Algorithm 6.1 with step (4) modified by choosing β_k as the first stationary point of $g(x)$ along the direction q_k.

Since $h(\hat{x}) = 0$ can not be guaranteed if a row vector of $J(\hat{x})$ is zero, some safeguards such as restarting strategy appear to be needed. However, our computational experience indicates that Algorithm 6.1 works very well, and the safeguards have not been used.

§7. Computational Results

To examine the approach proposed, some computational experiments have been done. By no means the best, the form of ECP-equation comprising ECP-direction matrix (3.10) and ECP-rate factor (3.13) is used as our test equation, i.e.,

$$\begin{cases} \dfrac{dx}{dt} = \varphi[c_2\beta P(\nabla^2 f)(P\nabla f) - (c_1 + c_2)(P\nabla f)] \equiv p(x), & \text{(7.1a)} \\ x(0) = x_0, & \text{(7.1b)} \end{cases}$$

where $\varphi = 1/\| \cdot \|_2$ and

$$\beta = \frac{(\nabla f)^T (P\nabla f)}{(P\nabla f)^T (\nabla^2 f)(P\nabla f)} . \tag{7.1c}$$

We take the ratio of c_2 to c_1, $DP = c_2/c_1 > 0$, rather than c_2 and c_1 separately, to specify concrete forms of (7.1). DP is referred to as direction parameter. After determination of a value of DP, the parameters c_2 and c_1 are then given through the following:

$$\begin{cases} c_1 = 1, \\ c_2 = DP, \end{cases} \text{ if } 0 \le DP \le 1; \quad \begin{cases} c_1 = 1/DP, \\ c_2 = 1, \end{cases} \text{ if } DP > 1. \tag{7.1d}$$

Note that (7.1) with $DP = 0$ has the same trajectory as Tanabe's equation that Brown and Bartholomew tested [2].

Model Algorithm 4.1 on equation (7.1), supported by Algorithms 5.1, 5.2 and 6.1, has been implemented in a double precision FORTRAN program on a FACOM 230 computer. Unfortunately, the author had not had at hand, for various reasons, the book by Hock and Schittkowski[5] then, and therefore had to construct by himself the five test problems which are collected in Appendix. These problems are casually formed, some of which are variants of the existing unconstrained ones.

Results of initial feasible points obtained via Algorithm 6.1 with $\varepsilon_0 = 10^{-5}$ are given in Table 1, where the procedure QUAD by Wolfe[17], with its (2.2.20) modified by using $h = \beta_{k-1}$, is utilized for line searches. In Table 1, NI is the number of iterations required and the index NC of computational work is defined by

$$NC = (NH + n\ NJ)m, \tag{7.2}$$

where NH is the number of evaluations of $h(x)$ and NJ is that of $J(x)$. The symbol MORM(H) denotes the value of $\|h(x_0)\|_2$. It should be indicated that the estimates $\hat{x}_0$, listed in Appendix, of initial points are also casually determined by the author himself; consequently the resulting initial points used are far away from respective solutions and more iterations are involved.

Table 1. Initial Point Results

Problem	N1	NH	NJ	NC	NORM(H)
TP1	1	6	1	10	0.36D−14
TP2	2	14	2	20	0.87D−06
TP3	7	69	7	194	0.26D−05
TP4	23	156	23	542	0.68D−05
TP5	14	89	14	477	Q.85D−05

In Algorithm 4.1, ε_1 is predetermined from ε_0 of Algorithm 6.1 through the formula below

$$\varepsilon_1 = \rho\,\varepsilon_0 \tag{7.3}$$

where the parameter ρ should be set to a positive number larger than 1, for it can not be expected that the precision with which the solution x^* to be located satisfies the constraints is higher than that of x_0. Seemingly appropriate, $\rho = 10$ is used in our program. Tables 2–6 contain results obtained via Algorithm 4.1 with $\alpha_0 = 0.05$ and $\varepsilon_1 = 10^{-4}$ on test problems TP1–5 respectively. In the program Criterion 4.1 with $\varepsilon = 10^{-6}$ and condition (4.8) with $N = 10^3$ are employed to terminate computations, and formula (5.2), Algorithms 5.1 and 5.2 are used for evaluations of the right-hand side $p(x)$ of (7.1).

In Tables 2–6, NR is the number of evaluations of $p(x)$, and the index of computational work NC is defined by

$$NC = NF + (NG + m\ NJ) \cdot n, \tag{7.4}$$

where $N.F$ and NG are the numbers of evaluations of the objective function f and its gradient ∇f respectively, and NJ is the same as in (7.2), i.e., the number of evaluations of $J(x)$. Besides, the symbols NORM(H), NORM(PG) and F denote, respectively, the values of $\|h\|_2$, $\|P\nabla f\|_2$ and f at x^*. By Theorem 3.5, the length of a trajectory can be given simply by summing up all the stepsizes in the numerical integrations. However, they are not listed explicitly in the Tables because numbers of iterations are found nearly to coincide with lengths of respective trajectories; and therefore NR can also be an index of lengths of trajectories.

Table 2. Results for TP1

DP	NR	NF	NC	NORM (H)	NORM (PG)	F
0.0	—	—	—	—	—	—
2.0	910	448	7732	0.22D−11	0.75D−02	0.3880748401D+03
4.0	633	310	5378	0.15D−11	0.42D−02	0.3880748401D+03
6.0	634	311	5387	0.15D−11	0.18D−02	0.3880748401D+03
8.0	529	259	4495	0.12D−11	0.15D−02	0.3880748401D+03
10.0	510	248	4332	0.12D−11	0.73D−02	0.3880748401D+03
12.0	514	251	4367	0.12D−11	0.12D−01	0.3880748401D+03
14.0	516	251	4383	0.12D−11	0.24D−02	0.3880748401D+03
16.0	500	243	4247	0.12D−11	0.10D−01	0.3880748401D+03
18.0	508	249	4317	0.12D−11	0.30D−02	0.3880748401D+03
20.0	516	251	4383	0.12D−11	0.39D−02	0.3880748401D+03

Table 3. Results for TP2

DP	NR	NF	NC	NORM(H)	NORM(PG)	F
0.0	—	—	—	—	—	—
2.0	207	97	1960	0.61D−06	0.13D−04	0.6891574534D+00
4.0	186	85	1759	0.83D−06	0.19D−04	0.6891576724D+00
6.0	195	91	1846	0.50D−06	0.12D−04	0.6891573382D+00
8.0	195	91	1846	0.24D−06	0.13D−04	0.6891570713D+00
10.0	203	95	1992	0.69D−06	0.79D−05	0.6891575269D+00
12.0	201	94	1903	0.84D−06	0.25D−05	0.6891576891D+00
14.0	204	95	1931	0.89D−06	0.28D−04	0.6891577362D+00
16.0	207	97	1960	0.92D−06	0.76D−05	0.6891577648D+00
18.0	200	93	1893	0.89D−06	0.40D−05	0.6891577349D+00
20.0	219	103	2074	0.89D−06	0.43D−05	0.6891577323D+00

Table 4. Results for TP3

DP	NR	NF	NC	NORM(H)	NORM(PG)	F
0.0	—	—	—	—	—	—
2.0	252	119	4151	0.26D−05	0.82D−02	0.3824632607D+03
4.0	216	101	3557	0.26D−05	0.82D−02	0.3824632590D+03
6.0	199	93	3277	0.26D−05	0.18D−02	0.3824632645D+03
8.0	195	92	3212	0.26D−05	0.36D−03	0.3824632599D+03
10.0	192	89	3161	0.26D−05	0.33D−02	0.3824632678D+03
12.0	212	101	3493	0.41D−05	0.60D−03	0.3824633081D+03
14.0	205	97	3377	0.44D−05	0.84D−03	0.3824633138D+03
16.0	200	93	3293	0.40D−05	0.24D−02	0.3824633068D+03
18.0	200	93	3293	0.35D−05	0.21D−02	0.3824632978D+03
20.0	200	94	3294	0.42D−05	0.89D−03	0.3824633108D+03

Table 5. Results for TP4

DP	NR	NF	NC	NORM(H)	NORM(PG)	F
0.0	—	—	—	—	—	—
2.0	—	—	—	—	—	—
4.0	375	181	7681	0.66D−05	0.11D−05	0.4387117574+00
6.0	389	189	7969	0.66D−05	0.12D−05	0.4387117476+00
8.0	400	193	8193	0.66D−05	0.45D−05	0.4387117464+00
10.0	416	202	8522	0.65D−05	0.20D−05	0.4387117485+00
12.0	419	202	8582	0.65D−05	0.43D−05	0.4387118043+00
14.0	430	208	8808	0.64D−05	0.81D−05	0.4387118330+00
16.0	441	214	9034	0.66D−05	0.49D−05	0.4387118571+00
18.0	450	218	9218	0.66D−05	0.81D−05	0.4387117555+00
20.0	442	214	9054	0.66D−05	0.92D−05	0.4387119027+00

Table 6. Results for TP5

DP	NR	NF	NC	NORM(H)	NORM(PG)	F
0.0	—	—	—	—	—	—
2.0	608	297	15497	0.83D−05	0.38D−03	0.1899872760+02
4.0	497	242	12667	0.80D−05	0.15D−03	0.1899872157+02
6.0	458	222	11672	0.83D−05	0.35D−03	0.1899872171+02
8.0	450	219	11469	0.83D−05	0.18D−03	0.1899872819+02
10.0	390	189	9939	0.84D−05	0.46D−03	0.1899872941+02
12.0	372	180	9480	0.84D−05	0.19D−03	0.1899872983+02
14.0	207	97	5272	0.82D−05	0.90D−04	0.1899873918+02
16.0	371	179	9454	0.11D−04	0.75D−04	0.1899873198+02
18.0	410	198	10448	0.11D−04	0.40D−03	0.1899872895+02
20.0	425	207	10832	0.12D−04	0.74D−04	0.1899873203+02

From 0 up to 20, eleven differing values of DP have been tested specifically with a view to determine the dependence of the performance of equation (7.1) upon the respective values of DP used. Under values of DP for which no data are listed convergence is still not attained after NR exceeds 10^3, whereas under those underlined the least computational work, indexed by NC, is required. The following can be drawn from Tables 2–6. For all of the problems, the worst results occur when $DP = 0$ corresponding to Tanabe's version that is among the ODE methods superior to some well-known SQP ones, according to [2]. In general, the lengths of trajectories get shorter and shorter, as values of DP increase from 0, until the best results, related to the shortest lengths of trajectories, are attained at some "moderate" values of DP; and after that the situation is reversed.

The worst performance of equation (7.1) with $DP = 0$ is not surprising indeed, for its discrete version is just the projected steepest descent method. The dependence of the performance of equation (7.1) upon the values of DP, as described in the last paragraph, is explained for the case of unconstrained minimization ($P = I$) by Pan[8], and might be considered similarly with the constrained case. This suggests that the value of DP should be determined in the light of the required accuracy of a solution and the extent of ill-conditioning of the encountered problem: larger value of DP is in general asked by higher accuracy and/or ill-conditioning. According to the author's experience, a suitable range of DP seems from 0 to 30, and a preferable value might be 10 or so. However, it is not yet clear how to gain an "optimal" value of DP beforehand.

Finally, we stress that like what reported by Brown and Bartholomew-Biggs[2], the author's computational experience equally indicates the remarkable robustness

of ODE approach, coinciding with the global analysis made in Section 2.

Acknowledgement. The author is indebted to Professor He Xu-chu for his supervision given at the Department of Mathematics, Nanjing University.

The author is also very grateful to Professors R. Tyrrell Rockafellar and Jim Burke for their valuable assistance.

Appendix : Test Problems

TP1. $n = 4, m = 1$ (Variant of Miele and Cantrell [6])

$$f = (\exp(x_1) - x_2)^4 + 100(x_2 - x_3)^6 + \mathrm{tg}^4(x_3 - x_4) + x_1^8,$$

$$h_1 = x_1 + 2(x_2 + x_3) + 2.1x_4 - 72;$$

$$\hat{x}_0 = (10, 10, 10, 10)^T,$$

$$x_0 = (\theta, \theta, \theta, \theta)^T, \text{ where } \theta = 10.14084507042254,$$

$$x^* = (2.00457, 10.67156, 11.35910, 12.34957)^T.$$

TP2. $n = 3, m = 1$

$$f = x_1^2 + 15.5x_2^2 + 2.5x_3^2,$$

$$h_1 = x_3 + x_1(x_1 + x_3) - 0.7\exp(x_2);$$

$$\hat{x}_0 = (-3, 1.5, 1.8)^T,$$

$$x_0 = (-2.755767454706105, 1.744233545293894, 2.044232736028758)^T,$$

$$x^* = (-0.82341, -0.02257, 0.03611)^T.$$

TP3. $n = 4, m = 2$ (Variant of Wood, quoted by Colville [18])

$$f = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2$$
$$+10.1[(x_2 - 1)^2 + (x_4 - 1)^2] + 19.8(x_2 - 1)(x_4 - 1),$$

$$h_1 = x_1 + 2(x_2 + x_3) + 3x_4 + x_1x_3x_4 - 52,$$

$$h_2 = x_4 - x_2^4 + 2;$$

$$\hat{x}_0 = (-2,\ 1.6,\ 0.5,\ -1)^T,$$

$$x_0 = (1.97958006498282, 1.5104718720025 12, 4.479587006498282,$$

$$3.205355006031058)^T,$$

$$x^* = (1.44879, 1.69\hat{7}15, 2.54165, 6.29624)^T.$$

TP4. $n = 5, m = 2$ (Variant of Miele et al. [7])

$$f = (x_1 - 1)^2 + (x_1 - x_2)^2 + (x_3 - 1)^2 + 13(x_4 - 1)^4 + 10(x_5 - 1)^6,$$

$$h_1 = x_1^2 x_4 + \sin(x_4 - x_5) - 2\sqrt{2},$$

$$h_2 = x_2 + x_4^2 x_3^4 - 8 - \sqrt{2};$$

$$\hat{x}_0 = (-0.5,\ -1,\ 1,\ 3,\ -0.8)^T,$$

$$x_0 = (0.9980086750148415, -0.9649589916751778, 1.035041008324820,$$

$$3.007227386646867, -0.3019913369060872)^T,$$

$$x^* = (1.33875, 1.36159, 1.49269, 1.27358, 0.69621)^T.$$

TP5. $n = 5, m = 3$ (Variant of Powell [10])

$$f = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4 + x_5^2,$$

$$h_1 = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10,$$

$$h_2 = x_2 x_3 - 5x_4 x_5 + x_4,$$

$$h_3 = x_1^3 + x_2^3 + 1;$$

$$\hat{x}_0 = (-1.7,\ 2.0,\ 2.0,\ -0.8,\ -1.0)^T,$$

$$x_0 = (-1.797000406080387, 1.687204801504492, 1.728714139965678,$$

$$-0.6176618997576491, -0.7444301413251398)^T,$$

$$x^* = (-1.00033, 0.09953, 0.03071, -0.00019, -2.99808)^T.$$

References

[1] Pan Ping-qi, New ODE Methods of Equality Constrained Optimization (I): Equations, *J. Comput. Math.*, **14** : 1 (1992), 77–92.

[2] A.A. Brown and M.C. Bartholomew-Biggs, ODE vs SQP methods for constrained optimization, The Numerical Optimization Center, Hatfield Polytechnic, Tech. Repor No.179, 1987.

[3] A.R. Colville, A comparative study of non-linear programming codes, IBM New York Scientific Center, Tech. Report 320–2925, 1968.

[4] R. W. Hamming, Stable predictor-corrector methods for ordinary differential equations, *J. Assoc. Comput. Machin.*, **14** (1959), 37–47.

[5] W. Hock and K. Schittkowski, Test examples for nonlinear programming codes, Lecture Notes in Economics and Mathematical Systems, No.187, Springer-Verlag, New York, 1981.

[6] A. Miele and J.W. Cantrel, Memory-Gradient Method for the Minimization of Functions, Symposium on Optimization, Nice., 1969.

[7] A. Miele, P.E. Moseley, A.V. Levy and G.M. Coggins, On the Method of Multipliers for Mathematical Programming Problems, *J. of Optimization Theory and Applications*, **10** (1972), 1–33.

[8] Pan Ping-qi, Differential equation methods for unconstrained optimization, *Numer. Math.*, **4** : 4 (1982), 338–349.

[9] E. Polak, Computational Methods in Optimization, Academic Press, New York, 1971.

[10] M.J.D. Powell, A Method for Nonlinear Constraints in Minimization Problems, in Optimization (Ed. R. Fletcher), Academic Press, London and New York, 1969, 283–298.

[11] M.J.D. Powell, Approximation Theory and Methods, Cambridge University Press, Cambridge, 1981, 212–215.

[12] K. Tanabe, A Geometric method in non-linear Programming, Brookhaven National Laboratory, New York, Technical Report 23343–AMD780, 1977.

[13] K. Tanabe, Differential geometric approach to extended grg methods with enforced feasibility in nonlinear programming: Global analysis, Brookhaven National Laboratory, New York, Technical Report 24497–AMD797, 1978.

[14] K. Tanabe, Differential geometric methods in nonlinear programming, Brookhaven Laboratory, New York. Technical Report 26730–AMD831, *Math. Software*, **10** : 3 (1979), 200–316.

[15] K. Tanabe, Differential geometric methods in nonlinear programming, Brookhaven Laboratory, New York, Technical Report 26730–AMD831, *Math. Software*, **10** : 3 (1979), 200–316.

[16] J.H. Wilkinson, Some Recent Advances in Numerical Linear Algebra, in The State of the Art in Numerical Analysis (Ed. D. Jacobs), Academic Press, London, 1979, 10–18.

[17] M.A. Wolfe, Numerical Methods for Unconstrained Optimization, Van Nostrand Reinhold Company, London, 1978.