Optimization Methods and Software, 1998, vol.10, pp. 125–146
Reprints available directly from the publisher
Photocopying permitted by license only

AUTOMATIC DIFFERENTIATION AND SPECTRAL PROJECTED GRADIENT METHODS FOR OPTIMAL CONTROL PROBLEMS

[a] ERNESTO G. BIRGIN[1] and [b] YURI G. EVTUSHENKO[2]

[a] *Department of Applied Mathematics, IMECC-UNICAMP, CP 6065, CEP 13081-970, Campinas, SP, Brazil;*
[b] *Computing Centre of Russian Academy of Sciences, 40 Vavilov Street, 117967 Moscow GSP-1, Russia*

Dedicated to Masao Iri on the occasion of his 65th birthday

(Revised 11 November 2003)

Automatic differentiation and non-monotone spectral projected gradient techniques are used for solving optimal control problems. The original problem is reduced to a nonlinear programming one using general Runge–Kutta integration formulas. Canonical formulas which use a fast automatic differentiation strategy are given to compute derivatives of the objective function. On the basis of this approach, codes for solving optimal control problems are developed and some numerical results are presented.

Keywords: Automatic differentiation, spectral projected gradient, non-monotone line search, optimal control problem, software for optimal control problems, Runge–Kutta integration methods

1 INTRODUCTION

Much attention has been paid to the development of numerical methods for solving optimal control problems. The most popular approach in this field turned out to be the reduction of the original problem to a nonlinear programming one (see, for example, [11, 29, 33, 34]). In [12], it was shown that the computation of the gradient in this particular case is closely related to the fast automatic differentiation (FAD) techniques (see [21, 22, 25, 26, 27]). In [13], using generalized FAD expressions, the exact gradient of the objective function of a general multi-step process was derived in a very simple canonical form. One of the aims of this paper is to show the application of these canonical formulas to optimal control processes being integrated by the Runge–Kutta family of numerical methods. There are many papers concerning numerical comparisons between automatic differentiation, finite differences and symbolic differentiation. See, for example, [1, 2, 6, 7, 21] among others.

Another objective is to test the behavior of the spectral projected gradient methods introduced in [5]. These methods combine the classical projected gradient with two recently developed ingredients in optimization:

[1]Sponsored by FAPESP (Grants 95/2452-6 and 97/12033-6).
[2]Sponsored by the Russian Foundation for Basic Research (Grants 96/15-96124 and 98/01-00517) and FAPESP (Grant 96/6631-5).

(i) the non-monotone line search schemes of Grippo et al. [24],

(ii) the spectral step-length (introduced by Barzilai and Borwein [3] and analyzed by Raydan [30, 31]).

This choice of the step-length requires little computational work and greatly speeds up the convergence of gradient methods. The numerical experiments presented in [5], showing the high performance of these fast and easily implementable methods, motivate us to combine the spectral projected gradient methods with automatic differentiation. Both tools are used in this work for the development of codes for numerical solution of optimal control problems.

In Section 2 of this paper, we apply the canonical formulas to the discrete version of the optimal control problem. In Section 3, we give a concise survey about spectral projected gradient algorithms. Section 4 presents some numerical experiments. Some final remarks are presented in Section 5.

2 CANONICAL FORMULAS

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

$$\frac{dx(t)}{dt} = f(x(t), u(t), \xi), \qquad T_0 \leq t \leq T_f, \tag{1}$$

where $x : [T_0, T_f] \to \mathbb{R}^{n_x}$, $u : [T_0, T_f] \to U \subseteq \mathbb{R}^{n_u}$, U compact, and $\xi \in V \subseteq \mathbb{R}^{n_\xi}$. Function x is called state, u is the control and ξ is the vector of design parameters. The solution of (1) is a function $x(t)$ with initial condition $x(T_0) = x_0$. In general, the scalars T_0, T_f and vector x_0 are fixed. If T_0, T_f or x_0 have to be optimized, then we can include them in the vector of design parameters ξ.

The problem is to find a control function $u(t) \in U$ and a vector of design parameters $\xi \in V$ that minimize a cost functional $W(T_0, T_f, x(T_f), u(T_f), \xi)$ subject to the mixed constraints on state, control and vector of design parameters:

$$h(x(t), u(t), \xi) = 0, \qquad q(x(t), u(t), \xi) \leq 0, \qquad T_0 \leq t \leq T_f. \tag{2}$$

As a rule, this problem is reduced to a mathematical programming one using a discretization scheme. The control function $u(t)$ is approximated by a piecewise constant function in which the accuracy of discretization depends on the problem to be solved. Sometimes this accuracy should be rather high and the software should permit us to provide it. Having some experience in solving practical problems, we came to the conclusion that very often the accuracy of the integration must be higher than the accuracy of the control function discretization. Therefore, for the sake of simplicity, it is possible to assume that the control vector u is constant in each interval of integration. Discretizing system (1), we obtain an N step process in which functions x and u are naturally represented as vectors,

$$x^\top = [x_0^\top, x_1^\top, \ldots, x_N^\top] \;\text{ and }\; u^\top = [u_0^\top, u_1^\top, \ldots, u_N^\top],$$

where $x_i = x(t_i) \in \mathbb{R}^{n_x}$, $u_i = u(t_i) \in \mathbb{R}^{n_u}$ and $t_i = T_0 + \sum_{k=0}^{i-1} \Delta t_k$ for $i = 0, 1, \ldots, N$, $0 < \Delta t_i \in \mathbb{R}$ are the discretization steps satisfying

$$\sum_{i=0}^{N-1} \Delta t_i = T_f - T_0, \tag{3}$$

$x \in \mathbb{R}^{n_x(N+1)}$ and $u \in \mathbb{R}^{n_u(N+1)}$. The discrete version of (1) is split into the N relations

$$x_i = F(X_i, U_i, \xi), \qquad i = 1, 2, \ldots, N, \tag{4}$$

where X_i and U_i are given sets of variables x_j and u_j, respectively, and the index j takes values from 0 to N. At the initial step we have $X_0 = U_0 = \emptyset$ and, for simplicity, we write $x_0 = F(X_0, U_0, \xi)$. The mixed constraints (2) are considered at each grid point as

$$h(x_i, u_i, \xi) = 0, \qquad q(x_i, u_i, \xi) \leq 0, \qquad i = 0, 1, \ldots, N, \tag{5}$$

and then the discretized optimal control problem, for an approximated solution of the original problem, can be written as

$$\begin{aligned} &\text{Minimize } W(T_0, T_f, x_N, u_N, \xi) \text{ subject to } u_i \in U, \quad \xi \in V, \\ &h(x_i, u_i, \xi) = 0, \qquad q(x_i, u_i, \xi) \leq 0. \end{aligned} \tag{6}$$

By fixing vectors u and ξ it is possible to obtain, from (4), the state vector $x_N(u, \xi)$, and define the composite function $\mathcal{W}(u, \xi) = W(T_0, T_f, x_N(u, \xi), u_N, \xi)$. For numerical minimization of this function, it is important to know the total derivatives of $\mathcal{W}$ with respect to u and ξ, since it allows us to use efficient gradient type minimization algorithms. In [13], it was shown that, for multistep processes like (4), formulas to compute the total derivatives $d\mathcal{W}/du$ and $d\mathcal{W}/d\xi$ can be obtained as follows.

For each set X_i and U_i we introduce the sets of indices Q_i and K_i containing the indices of all variables x_j and u_j belonging to the sets X_i and U_i, respectively, i.e.,

$$Q_i = \{j : x_j \in X_i\} \quad \text{and} \quad K_i = \{j : u_j \in U_i\}.$$

We also define their *conjugate* indices sets as

$$\bar{Q}_i = \{j : x_i \in X_j\} \quad \text{and} \quad \bar{K}_i = \{j : u_i \in U_j\}.$$

The sets Q_i and K_i are the *input* indices and $\bar{Q}_i$ and $\bar{K}_i$ are the *output* indices sets at the ith step. Let us introduce the adjoint vectors $p_i \in \mathbb{R}^{n_x}$ for $i = 0, 1, \ldots, N$, the total adjoint vector $p^\top = [p_0^\top, p_1^\top, \ldots, p_N^\top] \in \mathbb{R}^{n_x(N+1)}$ and the auxiliary function

$$E(x, u, \xi, p) = W(T_0, T_f, x_N, u_N, \xi) + \sum_{i=0}^{N} F(X_i, U_i, \xi)^\top p_i. \tag{7}$$

Therefore, the formulas for the computation of the adjoint vectors p_i and the total derivatives $d\mathcal{W}/du$ and $d\mathcal{W}/d\xi$ can be written in the following *canonical* form:

$$x_i = E_{p_i}(x, u, \xi, p) = F_{x_i}(X_j, U_j, \xi), \tag{8}$$

$$p_i = E_{x_i}(x, u, \xi, p) = W_{x_i}(T_0, T_f, x_N, u_N, \xi) + \sum_{j \in \bar{Q}_i} F_{x_i}(X_j, U_j, \xi)^\top p_j, \tag{9}$$

$$\frac{d\mathcal{W}(u, \xi)}{du_i} = E_{u_i}(x, u, \xi, p) = W_{u_i}(T_0, T_f, x_N, u_N, \xi) + \sum_{j \in \bar{K}_i} F_{u_i}(X_j, U_j, \xi)^\top p_j, \tag{10}$$

$$\frac{d\mathcal{W}(u, \xi)}{d\xi} = E_{\xi}(x, u, \xi, p) = W_{\xi}(T_0, T_f, x_N, u_N, \xi) + \sum_{j=0}^{N} F_{\xi}(X_j, U_j, \xi)^\top p_j, \tag{11}$$

for i varying from 0 to N. Here and from now on, Y_w denotes the partial derivative of function Y with respect to w, i.e., $Y_w = \partial Y / \partial w$ whereas dZ/dw denotes the total derivative of Z with respect to w. We assume that relation (8) defines an explicit process, i.e., at each step i the input set Q_i is such that for any $k \in Q_i$ the inequality $k < i$ holds. In this case, from (9), we have

$$p_N = W_{x_N}(T_0, T_f, x_N, u_N, \xi). \tag{12}$$

Note that, considering expression (9), we can conclude that the adjoint values p_i are the partial derivatives of $\mathcal{W}$ with respect to the state variables x_i.

As a practical application, we consider the multi-step process (4) given by the Runge–Kutta family methods defined by

$$\begin{cases} x_{i+1}^0 &= F(X_{i+1}, U_{i+1}, \xi) = x_i^0 + \Delta t_i \sum_{j=0}^{M-1} \alpha_j f(z_i^j), \\ x_i^{j+1} &= x_i^0 + \beta_j \Delta t_i f(z_i^j), \qquad j = 0, \dots, M-2, \end{cases} \tag{13}$$

for $i = 0, \dots, N-1$. Here, $x_i^0 \equiv x_i$, $t_i^0 \equiv t_i$, $t_i^j = t_i + \beta_{j-1}\Delta t_i$, $u_i^j = u(t_i^j)$, $z_i^j = (x_i^j, u_i^j, \xi)$ and $x_i^j \in \mathbb{R}^{n_x}$ are auxiliary vectors. Since we have assumed that the control variables u_i are constant in the integration step, we have $u_i = u_i^j$ and $z_i^j = (x_i^j, u_i, \xi)$. Table I shows some examples of the Runge–Kutta family methods (see, for example, [32]).

TABLE I. Examples of Runge–Kutta family methods

Integration method	M	α	β
Runge–Kutta order 1 or Euler	1	$\alpha_0 = 1$	
Runge–Kutta order 2 or modified Euler	2	$\alpha_0 = 0, \quad \alpha_1 = 1$	$\beta_0 = 0.5$
Runge–Kutta order 2	2	$\alpha_0 = \alpha_1 = 0.5$	$\beta_0 = 1$
Runge–Kutta order 4	4	$\alpha_0 = \alpha_3 = 1/6$	$\beta_0 = \beta_1 = 0.5$
		$\alpha_1 = \alpha_2 = 1/3$	$\beta_2 = 1$

Substituting (13) in (7) we obtain

$$\begin{aligned} E(x, u, \xi, p) = W(T_0, T_f, z_N^0) \quad &+ \quad \sum_{i=0}^{N-1} \sum_{j=0}^{M-2} [x_i^0 + \Delta t_i \beta_j f(z_i^j)]^\top p_i^{j+1} + \\ &+ \quad \sum_{i=0}^{N-1} \left[x_i^0 + \sum_{j=0}^{M-1} \Delta t_i \alpha_j f(z_i^j) \right]^\top p_{i+1}^0, \end{aligned} \tag{14}$$

and, applying the canonical formulas (8) – (11), we get

$$p_i^0 = W_{x_i^0}(T_0, T_f, z_N^0) + p_i^1 + \Delta t_i \beta_0 f_{x_i^0}(z_i^0)^\top p_i^1 + p_{i+1}^0 + \Delta t_i \alpha_0 f_{x_i^0}(z_i^0)^\top p_{i+1}^0, \tag{15}$$

$$p_i^j = W_{x_i^j}(T_0, T_f, z_N^0) + \Delta t_i \beta_j f_{x_i^j}(z_i^j)^\top p_i^{j+1} + \Delta t_i \alpha_j f_{x_i^j}(z_i^j)^\top p_{i+1}^0, \tag{16}$$

$$\frac{dE}{du_i} = W_{u_i}(T_0, T_f, z_N^0) + \sum_{j=0}^{M-2} \Delta t_i \beta_j f_{u_i}(z_i^j)^\top p_i^{j+1} + \sum_{j=0}^{M-1} \Delta t_i \alpha_j f_{u_i}(z_i^j)^\top p_{i+1}^0, \tag{17}$$

for $i = 0, 1, \dots, N-1$ and $j = 1, 2, \dots, M-1$, and

$$\frac{dE}{du_N} = W_{u_N}(T_0, T_f, z_N^0), \tag{18}$$

$$\frac{dE}{d\xi} = W_\xi(T_0, T_f, z_N^0) + \sum_{i=0}^{N-1} \sum_{j=0}^{M-2} \Delta t_i \beta_j f_\xi(z_i^j)^\top p_i^{j+1} + \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \Delta t_i \alpha_j f_\xi(z_i^j)^\top p_{i+1}^0. \tag{19}$$

Finally, rearranging formulas (15) – (19) and discarding the null derivatives, we arrive at Subroutine 2.1 for the computation of $\mathcal{W}(u,\xi)$, $d\mathcal{W}(u,\xi)/du$ and $d\mathcal{W}(u,\xi)/d\xi$. It is necessary to remark that the approach presented above (formulas (15) – (19)) is simpler and closer to the computer implementation than the analogous results given in [11, pp. 379–382]. Meanwhile, our formulas can be used for the whole class of Runge–Kutta methods. This improvement comes from the application of the auxiliary function (7) and the canonical formulas (8) – (11) introduced in [13].

Subroutine 2.1

Set $x_0^0 \leftarrow x_0$.
For $i = 0, \ldots, N-1$ (*increasing loop*)
    For $j = 0, \ldots, M-2$ (*increasing loop defined only for* $M \geq 2$)
        Set $y \leftarrow f(z_i^j)$,
        compute $x_i^{j+1} = x_i^0 + \Delta t_i \beta_j y$ and
        compute $x_{i+1}^0 = x_{i+1}^0 + \Delta t_i \alpha_j y$.
    endfor
    Set $y \leftarrow f(z_i^{M-1})$ and
    compute $x_{i+1}^0 = x_{i+1}^0 + \Delta t_i \alpha_{M-1} y$.
endfor
Compute $W(T_0, T_f, z_N^0)$.
Set $dE/d\xi \leftarrow W_\xi(T_0, T_f, z_N^0)$,
set $dE/du_N \leftarrow W_{u_N}(T_0, T_f, z_N^0)$ and
set $p_N \leftarrow W_{x_N}(T_0, T_f, z_N^0)$.
For $i = N-1, \ldots, 0$ (*decreasing loop*)
    Compute $dE/d\xi = dE/d\xi + \Delta t_i \alpha_{M-1} f_\xi(z_i^{M-1})^\top p_{i+1}$,
    compute $dE/du_i = \Delta t_i \alpha_{M-1} f_{u_i}(z_i^{M-1})^\top p_{i+1}$,
    compute $v = \Delta t_i \alpha_{M-1} f_{x_i^{M-1}}(z_i^{M-1})^\top p_{i+1}$ and
    compute $p_i = p_i + v$.
    For $j = M-2, \ldots, 0$ (*decreasing loop defined only for* $M \geq 2$)
        Compute $y = \alpha_j p_{i+1} + \beta_j v$,
        compute $dE/d\xi = dE/d\xi + \Delta t_i \alpha_j f_\xi(z_i^j)^\top y$,
        compute $dE/du_i = dE/du_i + \Delta t_i \alpha_j f_{u_i}(z_i^j)^\top y$,
        compute $v = \Delta t_i f_{x_i^j}(z_i^j)^\top y$ and
        compute $p_i = p_i + v$.
    endfor
endfor

In Subroutine 2.1, $y, v \in \mathbb{R}^{n_x}$ are auxiliary vectors for intermediate computations. Observe that there is no need to save adjoint values p_i^j of intermediate variables x_i^j, when $j \neq 0$. For this reason, we use notation p_i for adjoint values p_i^0 of x_i^0. This kind of implementation is a mixed strategy which tries to find an equilibrium between computational cost and memory storage. It is easy to see that, as W and its partial derivatives, W_ξ, W_{u_N} and W_{x_N}, are being computed together, we should call for a unique function which computes all of them using the reverse mode. This is not the case of function f and its derivatives, which are being computed at different steps. For this reason, it is not possible to take advantage of common expressions between f and f_{x_i}, f_{u_i} and f_ξ. We call this strategy *hybrid* FAD. In this implementation we are sacrificing computational time to save memory storage.

A particular and important class of control problems is when the goal is to minimize the duration of the process. A possible strategy to handle this situation is to introduce a new design parameter ξ^{n_ξ}, define the goal function in (6) as

$$W(T_0, T_f, z_N^0) = (T_f - T_0)\xi^{n_\xi}, \tag{20}$$

and rewrite (1) as

$$\frac{dx(t)}{dt} = \bar{f}(x, u, \xi) = \xi^{n_\xi} f(x, u, \xi), \qquad T_0 \le t \le T_f, \tag{21}$$

$$0 \le \xi^{n_\xi} < +\infty. \tag{22}$$

If we use Subroutine 2.1 to compute the gradient of (20), f will be computed twice at the same point: $\bar{f}(x, u, \xi) = \xi^{n_\xi} f(x, u, \xi)$ and $\bar{f}_{\xi^{n_\xi}}(x, u, \xi) = f(x, u, \xi)$. To overcome this problem, we should use, instead of the auxiliary vector $y \in \mathbb{R}^{n_x}$, a three-dimensional array $y \in \mathbb{R}^{n_x \times N \times M}$ (or NM vectors $y_i^j \in \mathbb{R}^{n_x}$). In this way the values of $f(z_i^j)$ will be saved in y_i^j and then used in the computation of $\bar{f}_{\xi^{n_\xi}}(z_i^j)$. This modification results from the observation of Subroutine 2.1 or from the application of the canonical formulas to the following reformulation of the Runge–Kutta family methods:

$$\begin{cases} f_i^j = f(z_i^j), & j = 0, \ldots, M-1, \\ x_{i+1}^0 = F(X_{i+1}, U_{i+1}, \xi) = x_i^0 + \Delta t_i \sum\limits_{j=0}^{M-1} [\alpha_j \xi^{n_\xi} f_i^j], & \\ x_i^{j+1} = x_i^0 + \beta_j \Delta t_i \xi^{n_\xi} f_i^j, & j = 0, \ldots, M-2, \end{cases} \tag{23}$$

for $i = 0, \ldots, N-1$. This is a special modification for a particular case. In this new approach we are doubling the storage space in order to avoid the computational cost of evaluating f twice. To show how to use the methodology, we will use Subroutine 2.1 for all the problems in the numerical experiments presented in Section 4.

3 NON-MONOTONE SPECTRAL PROJECTED GRADIENT METHODS

The non-monotone spectral gradient-projection algorithms have been introduced in [5]. These methods combine the classical projected gradient with two recently developed ingredients in optimization:

(i) the non-monotone line search schemes developed by Grippo *et al.* [24] for Newton's method and

(ii) the spectral step-length, introduced by Barzilai and Borwein [3] and analyzed by Raydan [30, 31].

In this section, we reproduce the definition of both versions of spectral projected gradient methods. The first one (SPG1 from now on) uses the classical projected gradient as a curvilinear search path. The second one (SPG2 from now on) uses, in order to avoid additional trial projections during the one-dimensional search process, the feasible spectral projected gradient as a search direction. These methods apply to problems like

$$\text{minimize } \varphi(x) \;\text{ subject to } x \in S, \tag{24}$$

where S is a closed convex set in $\mathbb{R}^n$. Throughout this section we assume that φ is defined and has continuous partial derivatives in an open set that contains S.

Given $z \in \mathbb{R}^n$, we define $P(z)$ as the orthogonal projection of z onto S. We denote $g(x) = \nabla\varphi(x)$. The algorithms use an integer $K \geq 1$, a small parameter $\lambda_{min} > 0$, a large parameter $\lambda_{max} > \lambda_{min}$, a sufficient decrease parameter $\gamma \in (0,1)$, and safeguarding parameters $0 < \sigma_1 < \sigma_2 < 1$. Given $\lambda_0 \in [\lambda_{min}, \lambda_{max}]$ and $x_0 \in S$, Algorithms 3.1 and 3.2 below, describe how to obtain x_* such that $\|P(x_* - g(x_*)) - x_*\|_2 = 0$.

Algorithm 3.1

Set $k \leftarrow 0$.
While $(\|P(x_k - g(x_k)) - x_k\|_2 \neq 0)$
    Compute $x_+ = P(x_k - \lambda_k g(x_k))$ and
    set $\alpha \leftarrow \lambda_k$.
    While $(\varphi(x_+) > \max_{0 \leq j \leq \min\{k, K-1\}} \{\varphi(x_{k-j})\} + \gamma\langle x_+ - x_k, g(x_k)\rangle)$
        define $\alpha \in [\sigma_1\alpha, \sigma_2\alpha]$ and
        compute $x_+ = P(x_k - \alpha g(x_k))$.
    endwhile
    Set $x_{k+1} \leftarrow x_+$ and
    compute $s_k = x_{k+1} - x_k$, $y_k = g(x_{k+1}) - g(x_k)$ and $b_k = \langle s_k, y_k\rangle$.
    If $(b_k \leq 0)$ set $\lambda_{k+1} \leftarrow \lambda_{max}$
    else compute $\lambda_{k+1} = \min\{\lambda_{max}, \max\{\lambda_{min}\langle s_k, s_k\rangle / b_k\}\}$.
    Set $k \leftarrow k+1$.
endwhile
Set $x_* \leftarrow x_k$.

Algorithm 3.2

Set $k \leftarrow 0$.
While $(\|P(x_k - g(x_k)) - x_k\|_2 \neq 0)$
    Compute $d_k = P(x_k - \lambda_k g(x_k)) - x_k$,
    compute $x_+ = x_k + d_k$ and
    set $\alpha \leftarrow 1$.
    While $(\varphi(x_{k+1}) > \max_{0 \leq j \leq \min\{k, K-1\}} \{\varphi(x_{k-j})\} + \gamma\alpha\langle d_k, g(x_k)\rangle)$
        define $\alpha \in [\sigma_1\alpha, \sigma_2\alpha]$ and
        compute $x_+ = x_k + \alpha d_k$.
    endwhile
    Set $x_{k+1} \leftarrow x_+$ and
    compute $s_k = x_{k+1} - x_k$, $y_k = g(x_{k+1}) - g(x_k)$ and $b_k = \langle s_k, y_k\rangle$.
    If $(b_k \leq 0)$ set $\lambda_{k+1} \leftarrow \lambda_{max}$
    else compute $\lambda_{k+1} = \min\{\lambda_{max}, \max\{\lambda_{min}, \langle s_k, s_k\rangle / b_k\}\}$.
    Set $k \leftarrow k+1$.
endwhile
Set $x_* \leftarrow x_k$.

The computation of $\alpha \in [\sigma_1\alpha, \sigma_2\alpha]$ uses a one-dimensional quadratic interpolation.

4 NUMERICAL RESULTS

Our code requires five subroutines implemented in **C/C++**. The first one, for the computation of the goal function W and its partial derivatives W_{x_N}, W_{u_N} and W_ξ, should use the reverse mode. The second one is used to compute the equality constraints h and h_{x_i}, h_{u_i} and h_ξ, the third one to compute the inequality constraints q and its partial derivatives; the fourth one computes f; finally, the last one computes f_{x_i}, f_{u_i} and f_ξ. Since in our test problems the functions (f and W) and the constraints (h and q) are simple enough, we decided to write their codes by hand. In general cases, automatic differentiation codes like **ADOL-C** [23] or **ADIC++** [6] should be used. Some parameters are connected to the Runge–Kutta methods and the integration step-size. In order to choose the Runge–Kutta method, there are input parameters M, α and β (see Table I). Methods listed in that table are only examples and any other method in the Runge–Kutta family can be used. Integration step-size Δt can be fixed or, as defined in (3), different integration step-sizes Δt_i can be considered.

The test problems are the ones reported in [20]. We divided the 11 problems into three groups according to their complexity. The first group includes the simplest control problems which do not have constraints on final state variables. Problems with final state constraints belong to the second group. Finally, the third group contains problems where the objective is to minimize the duration of the process and there are constraints at the final state. For the reproducibility of our results, and since the original reference is in Russian, all our final continuous formulations of the test problems are listed below. A full description of their physical meanings and optimal controls and trajectories can be found in [20]. It is necessary to remark that in this kind of optimization problem, which derives from an optimal control problem, the complexity of the goal function depends not only on the dimension of the solution, $n = n_u(N+1) + n_\xi$, but also on the dimensionality of state variables. A good measure of the complexity of the goal function is $n_{ocp} = (n_x + n_u)(N+1) + n_\xi$. Table II presents the groups and the variables n and n_{ocp}.

TABLE II. Problem sets

Group	Problem	n	n_{ocp}
1	1.a	201	804
	1.b	101	404
	1.c	101	404
	2	31	124
	3	161	483
2	4	41	123
	5	41	164
	6	162	567
3	7	21	85
	8	203	607
	9	163	568
	10	127	509
	11	161	484

Problem 1.a ($n_x = 3,\ n_u = 1,\ n_\xi = 0$).

$$f(x(t), u(t), \xi)^\top = [x^1(t), (1 - x^0(t)^2)x^1(t) - x^0(t) + u^0(t), x^0(t)^2 + x^1(t)^2 + u^0(t)^2], \quad T_0 \le t \le T_f,$$

$$x(T_0)^\top = [3, 0, 0], \quad T_0 = 0, \quad T_f = 10,$$
$$W(T_0, T_f, x(T_f), u(T_f), \xi) = x^2(T_f), \quad -10^{10} \le u^0(\cdot) \le 10^{10}$$
$$\text{and initial control} \quad u(\cdot) \equiv 0.$$

Problem 1.b ($n_x = 3,\ n_u = 1,\ n_\xi = 0$).

$$f(x(t), u(t), \xi)^\top = [x^1(t), (1 - x^0(t)^2)x^1(t) - x^0(t) + u^0(t), x^0(t)^2 + x^1(t)^2 + u^0(t)^2], \quad T_0 \le t \le T_f,$$
$$x(T_0)^\top = [0, 1, 0], \quad T_0 = 0, \quad T_f = 5,$$
$$W(T_0, T_f, x(T_f), u(T_f), \xi) = x^2(T_f), \quad -10^{10} \le u^0(\cdot) \le 10^{10}$$
$$\text{and initial control} \quad u(\cdot) \equiv 0.$$

Problem 1.c ($n_x = 3,\ n_u = 1,\ n_\xi = 0$).

$$f(x(t), u(t), \xi)^\top = [x^1(t), (1 - x^0(t)^2)x^1(t) - x^0(t) + u^0(t), x^0(t)^2 + x^1(t)^2 + u^0(t)^2], \quad T_0 \le t \le T_f,$$
$$x(T_0)^\top = [0, 1, 0], \quad T_0 = 0, \quad T_f = 5,$$
$$W(T_0, T_f, x(T_f), u(T_f), \xi) = x^2(T_f), \quad -0.8 \le u^0(\cdot) \le 0.8$$
$$\text{and initial control} \quad u(\cdot) \equiv 0.$$

Problem 2. Use of catalyst for two successive chemical reactions ($n_x = 2,\ n_u = 1,\ n_\xi = 0$).

$$f(x(t), u(t), \xi)^\top = [u^0(t)(10x^1(t) - x^0(t)), -u^0(t)(10x^1(t) - x^0(t)) - (1 - u^0(t))x^1(t)], \ T_0 \le t \le T_f,$$
$$x(T_0)^\top = [1, 0], \quad T_0 = 0, \quad T_f = 1,$$
$$W(T_0, T_f, x(T_f), u(T_f), \xi) = x^0(T_f) + x^1(T_f), \quad 0 \le u^0(\cdot) \le 1$$
$$\text{and initial control} \quad u(\cdot) \equiv 0.5.$$

Problem 3. Chemical reaction between gases ($n_x = 2$, $n_u = 1$, $n_\xi = 0$).

$$f(x(t), u(t), \xi)^\top = [f^0(x(t), u(t), \xi), f^1(x(t), u(t), \xi)],$$
$$f^0(x(t), u(t), \xi) = -2c^1 x^0(t) u^0(t) / (2c^0 + x^1(t)),$$
$$f^1(x(t), u(t), \xi) = -2f^1(x(t), u(t), \xi) - c^2 u^0(t)^2 (2x^1(t)/(2c^0 + x^1(t)))^2, \quad T_0 \le t \le T_f,$$
$$c^0 = 1.475 \times 10^{-2}, \quad c^1 = 1.8725688803 \times 10^7, \quad c^2 = 1.2162426427 \times 10^{14},$$
$$x(T_0)^\top = [0.0105, 0.0085], \quad T_0 = 0, \quad T_f = 8,$$
$$W(T_0, T_f, x(T_f), u(T_f), \xi) = -100x^1(T_f), \quad 0 \le u(\cdot) \le 1$$
$$\text{and initial control} \quad u(\cdot) \equiv 0.$$

Problem 4. Dolichobrachistochrone problem ($n_x = 2$, $n_u = 1$, $n_\xi = 0$).

$$f(x(t), u(t), \xi)^\top = \left[u^0(t), \sqrt{(1 + u^0(t)^2)/x^0(t)}\right], \qquad T_0 \le t \le T_f,$$

$$x(T_0)^\top = [3,0], \qquad T_0 = 0, \qquad T_f = 2,$$
$$W(T_0,T_f,x(T_f),u(T_f),\xi) = x^1(T_f), \quad x^0(T_f) = 10, \qquad -10^{10} \le u(\cdot) \le 10^{10}$$
$$\text{and initial control} \ \ u(\cdot) \equiv 5.$$

Problem 5. Rotation of an electrical machine ($n_x = 3$, $n_u = 1$, $n_\xi = 0$).

$$f(x(t),u(t),\xi)^\top = [x^1(t), -(4+0.8x^1(t)) + u^0(t), u^0(t)^2], \qquad T_0 \le t \le T_f,$$
$$x(T_0) = 0_{n_x}, \qquad T_0 = 0, \qquad T_f = 2,$$
$$W(T_0,T_f,x(T_f),u(T_f),\xi) = x^2(T_f), \qquad x^0(T_f) = 1, \quad x^1(T_f) = 0, \quad -10^{10} \le u(\cdot) \le 10^{10}$$
$$\text{and initial control} \ \ u(\cdot) \equiv 0.$$

Problem 6. Rocket motion ($n_x = 5$, $n_u = 2$, $n_\xi = 0$).

$$f(x(t),u(t),\xi)^\top = [x^1(t), u^0(t), x^3(t), u^1(t) - 9.81, u^0(t)^2 + u^1(t)^2], \qquad T_0 \le t \le T_f,$$
$$x(T_0)^\top = [-1,0,0,0,0], \qquad T_0 = 0, \qquad T_f = 4,$$
$$W(T_0,T_f,x(T_f),u(T_f),\xi) = \sqrt{x^4(T_f)}, \quad x^0(T_f) = x^1(T_f) = x^2(T_f) = x^3(T_f) = 0,$$
$$-10^{10}_{n_u} \le u(\cdot) \le 10^{10}_{n_u}$$
$$\text{and initial control} \ \ u(\cdot)^\top \equiv (0,1).$$

Problem 7. ($n_x = 3$, $n_u = 1$, $n_\xi = 1$).

$$f(x(t),u(t),\xi)^\top = \xi^0[x^1(t), x^2(t), u^0(t)], \qquad T_0 \le t \le T_f,$$
$$x(T_0)^\top = [16,0,0], \qquad T_0 = 0, \qquad T_f = 1,$$
$$W(T_0,T_f,x(T_f),u(T_f),\xi) = (T_f - T_0)\xi, \quad x(T_f) = 0_{n_x}, \quad -1 \le u(\cdot) \le 1, \qquad 0 \le \xi^0 \le 10^{10},$$
$$\text{initial control} \ \ u(\cdot) \equiv 0 \quad \text{and initial design parameter } \xi = 1.$$

Problem 8. Control of aircraft motion ($n_x = 4$, $n_u = 2$, $n_\xi = 1$).

$$f(x(t),u(t),\xi)^\top = \xi^0[x^2(t), x^3(t), -x^3(t) + u^0(t)\sin(u^1(t)), x^2(t) + u^0(t)\cos(u^1(t))], \ \ T_0 \le t \le T_f,$$
$$x(T_0)^\top = [10,0,0,0], \qquad T_0 = 0, \qquad T_f = 5,$$
$$W(T_0,T_f,x(T_f),u(T_f),\xi) = (T_f - T_0)\xi, \quad x(T_f) = 0_{n_x}, \quad 0 \le u^0(\cdot) \le 1,$$
$$-3.14 \le u^1(\cdot) \le 3.14, \qquad 0 \le \xi^0 \le 10^{10},$$
$$\text{initial control} \ \ u(\cdot) \equiv 0_{n_u} \ \ \text{and initial design parameter } \xi = 1.$$

Problem 9. Rocket dynamics with minimal duration ($n_x = 5$, $n_u = 2$, $n_\xi = 1$).

$$f(x(t),u(t),\xi)^\top = \xi^0[x^1(t), u^0(t), x^3(t), u^1(t) - 9.81, u^0(t)^2 + u^1(t)^2], \qquad T_0 \le t \le T_f,$$
$$x(T_0)^\top = [-1,0,1,0,0], \qquad T_0 = 0, \qquad T_f = 4,$$

$$W(T_0, T_f, x(T_f), u(T_f), \xi) = (T_f - T_0)\xi, \quad x^0(T_f) = x^1(T_f) = x^2(T_f) = x^3(T_f) = 0,$$

$$\sqrt{x^4(T_f)} - 13 = 0, \qquad -10^{10}_{n_u} \leq u(\cdot) \leq 10^{10}_{n_u}, \qquad 0 \leq \xi^0 \leq 10^{10},$$

$$\text{initial control} \;\; u(\cdot)^\top \equiv [0, 5] \;\; \text{and initial design parameter} \;\; \xi = 1.$$

Problem 10. Satellite launching ($n_x = 3$, $n_u = 1$, $n_\xi = 1$).

$$f(x(t), u(t), \xi)^\top = \xi^0[x^1(t), -x^0(t) + x^2(t), u^0(t)], \qquad T_0 \leq t \leq T_f,$$

$$x(T_0)^\top = [0.052, 0, -0.145], \qquad T_0 = 0, \qquad T_f = 6.28,$$

$$W(T_0, T_f, x(T_f), u(T_f), \xi) = (T_f - T_0)\xi, \quad x(T_f) = 0_{n_x}, \quad -0.3 \leq u(\cdot) \leq 0.3, \qquad 0 \leq \xi^0 \leq 10^{10},$$

$$\text{initial control} \;\; u(\cdot)^\top \equiv 0 \;\; \text{and initial design parameter} \;\; \xi = 1.$$

Problem 11. Stopping of a rotating body ($n_x = 2$, $n_u = 1$, $n_\xi = 1$).

$$f(x(t), u(t), \xi)^\top = \xi^0[x^1(t), u^0(t)], \qquad T_0 \leq t \leq T_f,$$

$$x(T_0)^\top = [2, 1], \qquad T_0 = 0, \qquad T_f = 8,$$

$$W(T_0, T_f, x(T_f), u(T_f), \xi) = (T_f - T_0)\xi, \quad x(T_f) = 0_{n_x}, \quad -1 \leq u(\cdot) \leq 1, \qquad 0 \leq \xi^0 \leq 10^{10},$$

$$\text{initial control} \;\; u(\cdot)^\top \equiv 0 \;\; \text{and initial design parameter} \;\; \xi = 1.$$

In the problem statements, v^i is the i-th scalar entrance of vector $v = [v^0, v^1, \ldots, v^{m-1}]^\top \in \in \mathbb{R}^m$ and c_m is a constant vector in $\mathbb{R}^m$ with all its entrances equal to $c \in \mathbb{R}$.

All the test problems have only equality constraints on final state variables x_N and box type constraints, $a \leq u \leq b$, with $a, b \in \mathbb{R}^{n_u(N+1)}$. In order to satisfy the equality constraints $h(x_N) = 0$, we applied the classical quadratic loss penalty function strategy: we added the term $\rho\|h(x_N)\|_2^2$ to the objective function of (6) and solved the box constrained subproblems

$$\text{minimize} \;\; W(T_0, T_f, x_N, u_N, \xi) + \rho\|h(x_N)\|_2^2,$$

for increasing values of $\rho \in \{10, 100, 1000, \ldots\}$ until we get $\|h(x)\|_2 \leq 10^{-5}$. In this case, the computation of h and the partial derivative h_{x_N} is included in the function for the computation of the penalized goal function. Since the bounds for the control variables are considered explicitly in the minimization process, the function for the computation of q, q_{x_i}, q_{u_i} and q_ξ is not necessary. Problems in group 3 were handled as suggested in (20) – (22). In this way, we have added the bound constraint (22) to these problems. Sometimes, to avoid local minimizers of the new design parameter ξ^{n_ξ}, with $\xi^{n_\xi} = 0$ (excluding the particular case in which the initial state satisfy the constraints on the final state), it is necessary to introduce a small constant $\delta \in \mathbb{R}$ and replace (22) by

$$0 < \delta \leq \xi^{n_\xi} < +\infty. \tag{25}$$

If $\xi^{n_\xi} = \delta$ at the solution, we decrease δ and start the minimization process again. We report the results of applying this strategy only once, with $\delta = 10^{-16}$ in Problem 9, and with $\delta = 0.2$ in Problems 10 and 11.

In order to discretize all the test problems, we have used the fourth-order Runge–Kutta method with integration step-size $\Delta t = 0.05$, except for Problem 2 where we have used $\Delta t =$

= 0.03. The stopping criterion was $\|g_P(x_k)\|_2 \leq \varepsilon$ with $\varepsilon = 10^{-6}$, $\varepsilon = 10^{-5}$, and $\varepsilon = 10^{-3}$, for Groups 1, 2 and 3, respectively.

All the experiments were run in a SPARCstation Sun Ultra 1, with an UltraSPARC 64 bits processor, 167-MHz clock and 128-MBytes of RAM memory. SPG codes are in C/C++ language and were compiled with g++ compiler (GNU project C and C++ compiler v2.7) using the optimization compiler option $-O4$. For the SPG methods, we used $\gamma = 10^{-4}$, $\lambda_{min} = 10^{-16}$, $\lambda_{max} = 10^{16}$ (except for Problem 1 where we used $\lambda_{max} = 10$), $\sigma_1 = 0.1$, $\sigma_2 = 0.9$ and $\lambda_0 = 1/\|g_P(x_0)\|_2$. In order to select the parameter K, we tested SPG2 in Problem 8, with K varying from 10 to 15. We chose $K = 13$ because it produced, for Problem 8, the solution with the smallest $\|h(x_N)\|_2$. However, several tests have shown that the solutions and the computational efforts do not change substantially for several values of K greater than 5.

Tables III and IV show, for problems in Groups 2 and 3, respectively, the performance of SPG1 for the solution of each subproblem. Tables V and VI show the same results for SPG2. We have reported the number of iterations (IT), the number of function-gradient evaluations (FGE), the number of line searches (LS), the CPU time in seconds (Time), the best function value ($\varphi(x)$), the quadratic loss ($\|h(x_N)\|_2^2$) and the 2-norm of the continuous projected gradient ($\|g_P(x)\|_2$). Finally, Tables VII and VIII summarize the performance of SPG methods for the complete set of problems.

In all cases, except for Problems 3 and 9, the solutions encountered coincide, with negligible differences, with solutions reported in [20]. The difference in the solution of Problem 3 is due to a possible print error in [20]. Another possibility for these differences is the choice of different integration methods and step-sizes. With only one exception, SPG methods found solutions with negligible differences in terms of function value. In Problem 9, SPG2 found the best solution with $\varphi(x) = 0.609295$, while SPG1 found a stationary point with $\varphi(x) = 1.705849$ (the same value as reported in [20]). Both methods satisfied the stopping criterion in all cases and there was no significant difference between their performances.

TABLE III. Intermediate solutions of SPG1 for problems of Group 2

Problem	ρ	IT	FGE	LS	Time	$\varphi(x)$	$\|h(x)\|_2^2$	$\|g_P(x)\|_2$
4	10^1	14	15	0	0.02	2.967166e+00	2.206227e-04	8.462335e-06
	10^2	1	4	1	0.00	2.971137e+00	2.204263e-06	8.020546e-06
	10^3	1	5	1	0.00	2.971534e+00	2.204067e-08	8.092395e-06
	10^4	1	6	1	0.00	2.971573e+00	2.204047e-10	8.100691e-06
	10^5	1	7	1	0.00	2.971577e+00	2.204045e-12	8.101532e-06
5	10^1	5	6	0	0.00	3.464720e+01	2.706461e-01	8.151036e-06
	10^2	7	8	0	0.00	3.965662e+01	3.123877e-03	1.687929e-07
	10^3	6	8	1	0.02	4.022061e+01	3.172508e-05	2.563081e-07
	10^4	6	9	1	0.00	4.027773e+01	3.177458e-07	3.601456e-07
	10^5	6	10	1	0.00	4.028345e+01	3.177954e-09	4.147059e-07
	10^6	6	11	1	0.00	4.028403e+01	3.178004e-11	4.738505e-07
6	10^1	9	10	0	0.05	1.961230e+01	6.234586e-04	6.384220e-07
	10^2	12	16	2	0.07	1.962353e+01	6.256737e-06	6.794644e-06
	10^3	12	17	2	0.07	1.962465e+01	6.258942e-08	7.262530e-06
	10^4	12	18	2	0.05	1.962477e+01	6.259195e-10	7.598892e-06
	10^5	12	19	2	0.08	1.962478e+01	6.259266e-12	8.043283e-06

TABLE IV. Intermediate solutions of SPG1 for problems of Group 3

Problem	ρ	IT	FGE	LS	Time	$\varphi(x)$	$\|h(x)\|_2^2$	$\|g_P(x)\|_2$
7	10^1	12092	16004	2715	8.58	7.941602e+0	2.906713e-03	7.837191e-4
	10^2	5404	7279	1163	3.85	7.994073e+0	2.970346e-05	9.570142e-4
	10^3	6800	9256	1521	4.97	7.999407e+0	2.974761e-07	7.868055e-4
	10^4	4123	5590	992	3.07	7.999964e+0	4.033835e-09	9.085447e-4
	10^5	2828	3880	746	2.08	8.000033e+0	4.317425e-11	9.928138e-4
8	10^1	341	456	60	3.72	1.020345e+1	4.286572e-03	5.798237e-4
	10^2	224	298	35	2.35	1.028079e+1	4.335458e-05	9.658054e-4
	10^3	53	63	7	0.50	1.028859e+1	4.342526e-07	9.960121e-4
	10^4	381	570	68	4.50	1.028937e+1	4.342951e-09	9.951110e-4
	10^5	48	62	7	0.53	1.028945e+1	4.343177e-11	9.984938e-4
9	10^1	252	488	59	1.85	1.697346e+0	4.244078e-04	7.124376e-4
	10^2	197	322	52	1.28	1.704994e+0	.324331e-06	2.836419e-4
	10^3	77	124	21	0.45	1.705763e+0	4.430674e-08	8.598459e-4
	10^4	168	269	42	1.03	1.705841e+0	4.455920e-10	7.390506e-4
	10^5	86	139	19	0.52	1.705849e+0	4.250948e-12	9.379793e-4
10	10^1	31	32	0	0.10	1.256000e+0	5.567425e-04	9.014210e-4
	10^2	20	33	5	0.12	1.256000e+0	5.311064e-04	8.818797e-4
	10^3	710	1265	198	3.80	1.418484e+0	3.304439e-05	9.977746e-4
	10^4	3439	6563	903	19.60	1.476615e+0	2.979178e-07	9.773216e-4
	10^5	2136	4221	577	12.55	1.481943e+0	2.957625e-09	9.617200e-4
	10^6	2682	5231	719	15.95	1.482473e+0	2.947335e-11	9.051665e-4
11	10^1	557	1046	150	2.80	4.095464e+0	3.271233e-03	8.446307e-4
	10^2	1005	1992	270	5.40	4.155792e+0	3.493484e-05	9.859478e-4
	10^3	928	1867	254	5.12	4.162059e+0	3.495614e-07	9.844122e-4
	10^4	998	1964	277	5.30	4.162686e+0	3.580196e-09	9.544532e-4
	10^5	475	910	130	2.48	4.162751e+0	3.621512e-11	9.980898e-4

TABLE V. Intermediate solutions of SPG2 for problems of Group 2

Problem	ρ	IT	FGE	LS	Time	$\varphi(x)$	$\|h(x)\|_2^2$	$\|g_P(x)\|_2$
4	10^1	14	15	0	0.02	2.967166e+00	2.206227e-04	8.462335e-06
	10^2	1	4	1	0.00	2.971137e+00	2.204263e-06	8.020546e-06
	10^3	1	5	1	0.02	2.971534e+00	2.204067e-08	8.092395e-06
	10^4	1	6	1	0.00	2.971573e+00	2.204047e-10	8.100691e-06
	10^5	1	7	1	0.02	2.971577e+00	2.204045e-12	8.101532e-06
5	10^1	5	6	0	0.00	3.464720e+01	2.706461e-01	8.151036e-06
	10^2	7	8	0	0.00	3.965662e+01	3.123877e-03	1.687929e-07
	10^3	6	8	1	0.00	4.022061e+01	3.172508e-05	2.563081e-07
	10^4	6	9	1	0.00	4.027773e+01	3.177458e-07	3.601456e-07
	10^5	6	10	1	0.00	4.028345e+01	3.177954e-09	4.147059e-07
	10^6	6	11	1	0.00	4.028403e+01	3.178004e-11	4.738505e-07
6	10^1	9	10	0	0.03	1.961230e+01	6.234586e-04	6.384220e-07
	10^2	12	16	2	0.07	1.962353e+01	6.256737e-06	6.794644e-06
	10^3	12	17	2	0.07	1.962465e+01	6.258942e-08	7.262531e-06
	10^4	12	18	2	0.08	1.962477e+01	6.259195e-10	7.598893e-06
	10^5	12	19	2	0.07	1.962478e+01	6.259266e-12	8.043284e-06

TABLE VI. Intermediate solutions of SPG2 for problems of Group 3

Problem	ρ	IT	FGE	LS	Time	$\varphi(x)$	$\|h(x)\|_2^2$	$\|g_P(x)\|_2$
7	10^1	10278	13437	2508	7.17	7.941438e+0	2.923215e-03	8.404570e-4
	10^2	8467	10812	1820	5.77	7.994078e+0	2.964787e-05	5.415757e-4
	10^3	8979	11478	2000	6.10	7.999407e+0	2.969850e-07	9.076313e-4
	10^4	4403	5711	1052	3.12	7.999974e+0	3.943387e-09	9.539757e-4
	10^5	2828	3725	724	1.97	8.000042e+0	4.423599e-11	7.832583e-4
8	10^1	578	748	128	5.87	1.020343e+1	4.288731e-03	4.883267e-4
	10^2	386	505	82	4.00	1.028079e+1	4.338337e-05	9.769583e-4
	10^3	177	230	35	1.82	1.028859e+1	4.339628e-07	4.413159e-4
	10^4	46	58	8	0.45	1.028937e+1	4.339369e-09	6.674503e-4
	10^5	44	59	8	0.45	1.028944e+1	4.342371e-11	5.973882e-4
9	10^1	13836	26253	3500	101.20	6.024155e-1	3.490189e-04	9.798078e-4
	10^2	178	338	51	1.37	6.086081e-1	3.423467e-06	8.323752e-4
	10^3	105	200	30	0.82	6.092257e-1	3.573099e-08	8.967019e-4
	10^4	169	330	56	1.33	6.092885e-1	3.762289e-10	9.619322e-4
	10^5	76	159	26	0.65	6.092949e-1	3.852315e-12	9.951342e-4
10	10^1	31	32	0	0.10	1.256000e+0	5.567425e-04	9.014210e-4
	10^2	18	23	4	0.07	1.256000e+0	5.319981e-04	8.823535e-4
	10^3	842	1338	238	4.03	1.418450e+0	3.310608e-05	9.916578e-4
	10^4	3628	6227	1049	18.82	1.476621e+0	2.980395e-07	9.211830e-4
	10^5	2733	4749	791	14.40	1.481954e+0	2.920001e-09	9.487892e-4
	10^6	2050	3598	589	11.05	1.482480e+0	2.935649e-11	9.673148e-4
11	10^1	1102	1915	347	4.93	4.095515e+0	3.288448e-03	7.641996e-4
	10^2	1345	2560	449	6.53	4.155787e+0	3.492347e-05	9.883468e-4
	10^3	1144	2204	343	5.67	4.162045e+0	3.526807e-07	9.583805e-4
	10^4	1277	2567	415	6.55	4.162674e+0	3.539575e-09	9.759158e-4
	10^5	1420	2825	433	7.20	4.162737e+0	3.505302e-11	9.864841e-4

TABLE VII. Performance of SPG1

Problem	IT	FGE	LS	Time	$\varphi(x)$	$\|g_P(x)\|_2$
1.a	434	558	77	2.62	2.141775e+01	2.332212e-07
1.b	87	95	6	0.20	2.621363e+00	6.558450e-07
1.c	43	44	0	0.12	4.340875e+00	4.100793e-07
2	48	50	1	0.02	9.519459e-01	7.612754e-07
3	1	19	1	0.08	-1.958250e+00	4.569610e-08
4	18	37	4	0.02	2.971577e+00	8.101532e-06
5	36	52	4	0.02	4.028403e+01	4.738505e-07
6	57	80	8	0.32	1.962478e+01	8.043283e-06
7	31249	42009	7137	22.55	8.000033e+00	9.928138e-04
8	1047	1449	177	11.60	1.028945e+01	9.984938e-04
9	780	1342	193	5.13	1.705849e+00	9.379793e-04
10	9018	17345	2402	52.12	1.482473e+00	9.051665e-04
11	3963	7779	1081	21.10	4.162757e+00	9.986710e-04

TABLE VIII. Performance of SPG2

Problem	IT	FGE	LS	Time	$\varphi(x)$	$\|g_P(x)\|_2$
1.a	500	654	96	3.43	2.141775e+01	3.202524e-07
1.b	87	95	6	0.22	2.621363e+00	6.557571e-07
1.c	43	44	0	0.12	4.340875e+00	4.100793e-07
2	49	52	2	0.03	9.519459e-01	6.363696e-07
3	1	12	1	0.08	-1.997609e+00	4.456312e-08
4	18	37	4	0.06	2.971577e+00	8.101532e-06
5	36	52	4	0.00	4.028403e+01	4.738505e-07
6	57	80	8	0.32	1.962478e+01	8.043284e-06
7	34955	45163	8104	24.13	8.000042e+00	7.832583e-04
8	1231	1600	261	12.59	1.028944e+01	5.973882e-04
9	14364	27280	3663	105.37	6.092949e-01	9.951342e-04
10	9302	15967	2671	48.47	1.482480e+00	9.673148e-04
11	6288	12071	1987	30.88	4.162737e+00	9.812947e-04

5 CONCLUSIONS

In this work we have shown how to apply the methodology introduced in [13] to the Runge–Kutta family of integration methods. An equivalent approach can be applied to other integration methods like, for example, Newton–Cotes and Adams–Moulton (see [32] and its numerous references [8, 14, 15, 16, 35]). The same remark must be made with respect to the optimization techniques. Instead of SPG methods, Subroutine 2.1 can be combined with other optimization algorithms like TNBOX [17] and LANCELOT [9], based on trust regions and truncated Newton approach. See [10] for a comparison between these two codes for box constrained minimization and see [5] for a comparison between TNBOX and SPG methods. See also [31] for a comparison between spectral gradient and conjugate gradient methods for unconstrained optimization. Moreover, instead of considering penalty function methods, many other nonlinear programming methods can be used (augmented Lagrangian, linearization, Newton like methods, interior point techniques, etc.). There are many ways of taking constraints (5) into account. If we use sequential minimization techniques (as penalty function methods), part of these constraints (for example, box constraints) can be considered explicitly in the optimization process, while other constraints may be penalized. It is worth to mention that, in all cases above, the auxiliary function (7) and the canonical formulas (8) – (11) can be used for computing total derivatives and derivatives of any order.

We tested the performance of SPG methods for the solution of optimal control problems because, as it is said in [5], “It is quite surprising that such a simple tool can be competitive with rather elaborated algorithms which use extensively tested subroutines and numerical procedures”. As our experimental results show, spectral projected gradient methods and automatic differentiation are, in fact, very useful tools for solving optimal control problems.

Finally, we have presented a set of problems and detailed information about the performance of SPG methods. This information can be used for future comparisons in the development of new software for solving optimal control problems. All the codes used can be requested by e-mail.

Acknowledgements

The authors are thankful to Lucio Tunes Santos for the careful reading of the manuscript.

References

[1] B.M. Averick, J.J. Moré, C.H. Bischof, A. Carle and A. Griewank, Computing large sparse Jacobian matrices using automatic differentiation, *SIAM Journal in Scientific Computing*, **15** (1994), 285–294.

[2] M.C. Bartholomew-Biggs, L. Bartholomew-Biggs and B. Christianson, Optimization & automatic differentiation in ADA: some practical experience, *Optimization Methods and Software*, **4** (1994), 47–73.

[3] J. Barzilai and J.M. Borwein, Two point step size gradient methods, *IMA Journal of Numerical Analysis*, **8** (1988), 141–148.

[4] D. Bertsekas, On the Goldstein–Levitin–Polyak gradient projection method, *IEEE Transactions on Automatic Control*, **21** (1976), 174–184.

[5] E.G. Birgin, J.M. Martínez and M. Raydan, Nonmonotone spectral projected gradient methods on convex sets, *RP 92/97* (Applied Mathematics Department, IMECC-UNICAMP, Brazil, 1997).

[6] E.G. Birgin, Automatic differentiation and applications, PhD Thesis (Applied Mathematics Department, IMECC-UNICAMP, Brazil, 1998) (in Portuguese).

[7] C.H. Bischof and A. Griewank, ADIFOR: A Fortran system for portable automatic differentiation, *Preprint MCS-P317-0792* (Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1992).

[8] J.C. Butcher, On Runge–Kutta processes of high order, *Journal of Australian Mathematic Society*, **4** (1964), 179–194.

[9] A.R. Conn, N.I.M. Gould and Ph.L. Toint, Global convergence of a class of trust region algorithms for optimization with simple bounds, *SIAM Journal on Numerical Analysis*, **25** (1988), 433–460. See also *SIAM Journal on Numerical Analysis*, **26** (1989), 764–767.

[10] M.A. Diniz–Ehrhardt, M.A. Gomes–Ruggiero and S.A. Santos, Comparing the numerical performance of two trust-region algorithms for large-scale bound-constrained minimization, *International Workshop of Numerical Linear Algebra and Optimization* (R.J.B. Sampaio and J.Y. Yuan, Department of Mathematics, Federal University of Paraná, Brazil, 1997), 23–24. (To appear in *Investigatión Operativa*).

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

[12] Y.G. Evtushenko, Automatic differentiation viewed from optimal control theory, *Automatic Differentiation of Algorithms. Theory, Implementation and Application* (A. Griewank and G.F. Corliss, *SIAM*, Philadelphia, 1997), pp. 25–30.

[13] Y.G. Evtushenko, Computation of exact gradients in distributed dynamic systems, *Optimization Methods and Software*, **9** (1998), 45–75.

[14] E. Fehlberg, New high-order Runge–Kutta formulas with stepsize control for systems of first and second-order differential equations, *Z. Angew. Math. Mech.*, **44** (1964), T17–T29.

[15] E. Fehlberg, New high-order Runge–Kutta formulas with an arbitrary small truncation error, *Z. Angew. Math. Mech.*, **46** (1964), 1–16.

[16] E. Fehlberg, Klassische Runge–Kutta formeln fünfter und siebenter ordnung mit schrittweitenkontrolle, *Computing*, **4** (1964), 93–106.

[17] A. Friedlander, J.M. Martínez and S.A. Santos, A new trust region algorithm for bound constrained minimization, *Applied Mathematics and Optimization*, **30** (1994), 235–266.

[18] A.A. Goldstein, Convex programming in Hilbert space, *Bulletin of the American Mathematical Society*, **70** (1964), 709–710.

[19] N.I. Grachev and Y.G. Evtushenko, A library of programs for solving optimal control problems, *USSR Computational Mathematics and Mathematical Physics*, **19** (1980), 99–119.

[20] N.I. Grachev and A.N. Filkov, *Solution of optimal control problems in system DISO* (Computing Center of Russian Academy of Sciences, Moscow, 1986) (in Russian).

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

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

[23] A. Griewank, ADOL-C. A package for the automatic differentiation of algorithms written in C/C++, *ACM TOMS*, **22** (1996), 131–167.

[24] L. Grippo, F. Lampariello and S. Lucidi, A nonmonotone line search technique for Newton's method, *SIAM Journal on Numerical Analysis*, **23** (1986), 707–716.

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

[26] M. Iri and K. Kubota, Methods of fast automatic differentiation and applications, *Research memorandum RMI* (Department of Mathematical Engineering and Instrumental Physics, Faculty of Engineering, University of Tokyo, 1987), pp. 87–102.

[27] M. Iri, History of automatic differentiation and rounding error estimation, *Automatic Differentiation of Algorithms. Theory, Implementation and Application* (A. Griewank and G.F. Corliss, *SIAM*, Philadelphia, 1991), pp. 3–16.

[28] E.S. Levitin and B.T. Polyak, Constrained minimization problems, *USSR Computational Mathematics and Mathematical Physics*, **6** (1966), 1–50.

[29] E. Polak, *Computation Methods in Optimization* (Academic Press, New York and London, 1971).

[30] M. Raydan, On the Barzilai and Borwein choice of steplength for the gradient method, *IMA Journal of Numerical Analysis*, **13** (1993), 321–326.

[31] M. Raydan, The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem, *SIAM Journal on Optimization*, **7** (1997), 26–33.

[32] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis* (Springer-Verlag Inc., Berlin, Heidelberg and New York, 1972).

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

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

[35] E.B. Shanks, Solution of differential equations by evaluation of functions, *Mathematical Computation*, **20** (1966), 21–38.