Journal of Process Control, Volume 15, Issue 8, December 2005, Pages 851-858.

A Formulation of Nonlinear Model Predictive Control Using Automatic Differentiation

Yi Cao

School of Engineering, Cranfield University, Bedford MK43 0AL, UK
Tel: (44)1234 750 111, Fax: (44)1234 750 728

Abstract

An efficient algorithm is developed to alleviate the computational burden associated with Nonlinear Model Predictive Control (NMPC). The new algorithm extends an existing algorithm for solutions of dynamic sensitivity from autonomous to non-autonomous differential equations using the Taylor series and automatic differentiation (AD). A formulation is then presented to recast the NMPC problem as a standard nonlinear programming problem by using the Taylor series and AD. The efficiency of the new algorithm is compared with other approaches via an evaporation case study. The comparison shows that the new algorithm can reduce computational time by two orders of magnitude.

Key words: Predictive Control, Optimal Control, Dynamic Sensitivity

1 Introduction

In the last two decades, linear model predictive control has been well recognized by industry due to its intuitiveness and capability to handle multivariable constraints. However, the extension to nonlinear model based predictive control (NMPC) has not been so successful although a significant amount of research effort has been put into this area. One of the main obstacles, which blocks NMPC techniques to become widely applicable, is the computational burden associated with the requirement to solve a set of nonlinear differential equations and a nonlinear dynamic optimization problem in real-time.

The objective of NMPC is to determine a set of future control moves in order to minimize a cost function based on a desired output trajectory over a prediction

Email address: y.cao@cranfield.ac.uk (Yi Cao).

horizon. The computation involved in solving the optimization problem at every sampling time can become so intensive, particularly for high-dimensional systems, that it could make on-line applications almost impossible [1]. There exist a number of strategies for tracking the optimal control problem through nonlinear programming (NLP) [2]: successive linearization, direct single and multiple shooting methods, and others. To efficiently solve the NLP problem derived, all these approaches require intensive computation of derivatives. In a typical situation, calculating dynamic sensitivity could take more than 70 percent of the total computation time for NLP. Hence, dynamic sensitivity calculation is the computational bottleneck of solving a dynamic optimization problem. There are three ways to calculate sensitivity of a dynamic system [3]: perturbations, sensitivity equations and adjoint equations. In a perturbation approach finite differences are used to approximate derivatives. Hence it needs at least applying N perturbations to the dynamic system to get the solution of a N-parameter sensitivity problem [3]. Alternatively, sensitivity can also be obtained by simultaneously solving the original ordinary differential equations (ODE) together with nN sensitivity equations, where n is the number of states [4]. Finally, sensitivity can be calculated by solving n adjoint equations (in reverse direction). A number of efficient solvers have been developed to tackle the dynamic sensitivity problem, for example, the CVODES package [5]. Recently, automatic differentiation (AD) techniques have been applied to solve dynamic optimization problems [6]. In previous work [7], a first-order approximation is introduced to simply the dynamic sensitivity equation by using AD so that the computation efficiency is improved. A similar approach has been proposed in [8] without using AD. However, due to the first-order approximation, the sensitivity obtained may not be accurate enough in some cases, particularly for NMPC with process constraints. In most published work using AD for dynamic optimization, AD has only been used to generate low (first and/or second) order derivatives, therefore efficiency of these approaches is not satisfactory.

In this work, the advantages of AD techniques have been intensively utilized to improve the efficiency of NMPC. More specifically, an existing algorithm to solve ODE and sensitivity using high-order Taylor series and AD for autonomous systems is extended to non-autonomous systems. An approach to estimate and control the error due to truncation of the Taylor series is also provided. Then, based on this algorithm, the NMPC problem has been reformulated as a NLP problem so that it can be efficiently solved by any modern NLP solvers. The paper is organized as follows. After a brief overview of AD, its principles to solve autonomous ODE's and to calculate dynamic sensitivity are explained in section 2. Section 3 extends the techniques to non-autonomous systems. Then, a formulation of NMPC using AD is proposed in section 4, where the issues of error analysis and control are also addressed. A case study is presented in section 5 to show the usage and efficiency of the new algorithm. Finally, the paper is concluded in section 6.

2 Automatic Differentiation

AD is a class of computational techniques for evaluating derivatives of functions defined in computer programs [9]. It is superior to other two approaches: symbolic differentiation and finite difference approximation. To compute derivatives symbolically using computer algebra software such as Mathematica or Maple, an enormous expression growth normally occurs due to a repeated evaluation of common sub-expressions. On the other hand, with finite difference approximation, accuracy of derivatives is restricted because of cancellation and truncation errors, particularly, for high order derivatives. Automatic differentiation techniques overcome these drawbacks by systematically applying the chain rule to functions defined by arbitrary computer programs. A computer program is equivalent to a computational graph consisting of a sequence of elementary operations whose derivatives are well known. Hence, by numerically applying the chain rule to these arithmetic sequences, not only can AD deliver truncation-error free derivatives but it also avoids code growth.

2.1 Taylor Series by AD

Consider a d-time continuously differentiable function, $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$. Let $\mathbf{x}(t) \in \mathbb{R}^n$ be given by the truncated Taylor series: $\mathbf{x}(t) = \mathbf{x}_{[0]} + \mathbf{x}_{[1]}t + \cdots + \mathbf{x}_{[d]}t^d$, with coefficients $\mathbf{x}_{[i]} = (i!)^{-1}(\partial^i \mathbf{x}(t)/\partial t^i)|_{t=0} \in \mathbb{R}^n$. Then, $\mathbf{z}(t) = \mathbf{f}(\mathbf{x}(t)) \in \mathbb{R}^m$ can be expressed by a Taylor expansion: $\mathbf{z}(t) = \mathbf{z}_{[0]} + \mathbf{z}_{[1]}t + \cdots + \mathbf{z}_{[d]}t^d + \mathcal{O}(t^{d+1})$ where $\mathbf{z}_{[j]} = (j!)^{-1}(\partial^j \mathbf{z}(t)/\partial t^j)|_{t=0} \in \mathbb{R}^m$. From the chain rule, $\mathbf{z}_{[j]}$ is uniquely determined by the coefficient vectors, $\mathbf{x}_{[i]}$ with $i \leq j$, *i.e.*

$$\mathbf{z}_{[j]} \equiv \mathbf{z}_{[j]}(\mathbf{x}_{[0]}, \mathbf{x}_{[1]}, \ldots, \mathbf{x}_{[j]}) \tag{1}$$

Nevertheless, inherently, functions $\mathbf{z}_{[j]}$ are also d-time continuously differentiable and their derivatives satisfy the identity [10]:

$$\frac{\partial \mathbf{z}_{[j]}}{\partial \mathbf{x}_{[i]}} = \frac{\partial \mathbf{z}_{[j-i]}}{\partial \mathbf{x}_{[0]}} := \mathbf{A}_{[j-i]} \equiv \mathbf{A}_{[j-i]}(\mathbf{x}_{[0]}, \mathbf{x}_{[1]}, \ldots, \mathbf{x}_{[j-i]}) \tag{2}$$

where, $\mathbf{A}_{[j]} \in \mathbb{R}^{n \times n}$, $j = 0, \ldots, d$ are also the Taylor coefficients of the Jacobian path, *i.e.* $\mathbf{f}'(\mathbf{x}(t)) = \mathbf{A}_0 + \mathbf{A}_1 t + \cdots + \mathbf{A}_d t^d + \mathcal{O}(t^{d+1})$.

AD techniques provide an efficient way to calculate these coefficient vectors, $\mathbf{z}_{[j]}$ and matrices, $\mathbf{A}_{[i]}$ [11]. For example, with the software package, ADOL-C [12], by using the forward mode of AD, all Taylor coefficient vectors for a given degree, d can be calculated simultaneously, whilst the matrices, $\mathbf{A}_{[i]}$ can be obtained by using the reverse mode of AD. The run time and memory requirement associated with these calculations grow only in a order of d^2.

2.2 Autonomous Differential Equation

When the above approach is applied to an autonomous differential equation, *i.e.* $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t))$, since $\mathbf{x}_{[k+1]} = \mathbf{z}_{[k]}/(k+1)$, all Taylor coefficients of $\mathbf{x}(t)$ up to any order can be iteratively obtained from $\mathbf{x}_{[0]} = \mathbf{x}(0)$ by using (1) [13]. Moreover, the sensitivity of Taylor coefficients against the initial value $\mathbf{x}_{[0]}$ can also be efficiently obtained by matrix accumulation from (2):

$$\mathbf{B}_{[k]} := \frac{d\mathbf{x}_{[k]}}{d\mathbf{x}_{[0]}} = \frac{1}{k}\frac{d\mathbf{z}_{[k-1]}}{d\mathbf{x}_{[0]}} = \frac{1}{k}\sum_{j=0}^{k-1}\frac{\partial \mathbf{z}_{[k-1]}}{\partial \mathbf{x}_{[j]}}\frac{d\mathbf{x}_{[j]}}{d\mathbf{x}_{[0]}} = \frac{1}{k}\sum_{j=0}^{k-1}\mathbf{A}_{[k-j-1]}\mathbf{B}_{[j]} \tag{3}$$

where $\mathbf{B}_{[k]} \in \mathbb{R}^{n\times n}$, $k = 0, \ldots, d$ are the Taylor coefficients of the solution to the sensitivity equations, $\dot{\mathbf{B}} = \mathbf{f}'(\mathbf{x})\mathbf{B}$, $\mathbf{B}_{[0]} = \mathbf{B}(0) = \mathbf{I}$.

3 Non-autonomous Systems

Although the above algorithm is very efficient, to make it applicable for NMPC, the algorithm has to be extended to solving dynamic sensitivity of non-autonomous state space systems:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), & \mathbf{x}(0) &= \mathbf{x}_{[0]} \\ \mathbf{y}(t) &= \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t)), & 0 &\le t \le h \end{aligned} \tag{4}$$

where, $\mathbf{u}(t) \in \mathbb{R}^m$ is the control input and $\mathbf{y}(t) \in \mathbb{R}^p$ the output. It is a normal practice, for example in [14], to convert the system (4) to autonomous by augmenting it with $\dot{\mathbf{u}} = 0$ so that the results described in the previous section can be directly used. However, the augmented system has m extra differential equations, hence the algorithm is not efficient particularly when m is large. In this work, an efficient approach is to be described as follows.

Using normalized time, $\tau = t/h$, the right-hand-side of the state equation becomes $\mathbf{z}(\mathbf{x}(\tau), \mathbf{u}(\tau)) := h\mathbf{f}(\mathbf{x}(\tau), \mathbf{u}(\tau))$ and the solution interval is $0 \le \tau \le 1$. Assume $\mathbf{u}(\tau) = \mathbf{u}_{[0]} + \mathbf{u}_{[1]}\tau + \cdots + \mathbf{u}_{[r]}\tau^r$, $r \le d$ and all its coefficients, $\mathbf{u}_{[k]}$, $k = 1, \ldots, r$ are known. Let $\mathbf{v} = \begin{bmatrix}\mathbf{u}_{[0]}^T & \cdots & \mathbf{u}_{[r]}^T\end{bmatrix}^T$. Using AD, the Taylor coefficients of $\mathbf{x}(\tau)$ and $\mathbf{y}(\tau)$ can be iteratively determined from $\mathbf{x}_{[0]}$ and $\mathbf{v}$.

$$\mathbf{x}_{[k+1]} = \frac{1}{k+1}\mathbf{z}_{[k]}(\mathbf{x}_{[0]}, \ldots, \mathbf{x}_{[k]}, \mathbf{v}), \qquad k = 0, \ldots, d-1 \tag{5}$$

$$\mathbf{y}_{[k]} = \mathbf{y}_{[k]}(\mathbf{x}_{[0]}, \ldots, \mathbf{x}_{[k]}, \mathbf{v}), \qquad k = 0, \ldots, d \tag{6}$$

Then, by applying AD to (5) and (6), the partial derivatives are obtained and

partitioned as follows:

$$\mathbf{A}_{[k]} = \left[\mathbf{A}_{[k]} \middle|\ \mathbf{A}_{[k]v}\right] := \left[\frac{\partial \mathbf{z}_{[k]}}{\partial \mathbf{x}_{[0]}} \middle|\ \frac{\partial \mathbf{z}_{[k]}}{\partial \mathbf{v}}\right], \qquad k = 0, \ldots, d-1 \tag{7}$$

$$\mathbf{C}_{[k]} = \left[\mathbf{C}_{[k]x} \middle|\ \mathbf{C}_{[k]v}\right] := \left[\frac{\partial \mathbf{y}_{[k]}}{\partial \mathbf{x}_{[0]}} \middle|\ \frac{\partial \mathbf{y}_{[k]}}{\partial \mathbf{v}}\right], \qquad k = 0, \ldots, d \tag{8}$$

The total derivatives are accumulated from these partial derivatives as follows:

$$\begin{aligned}\mathbf{B}_{[k]} &= \left[\mathbf{B}_{[k]x} \mid \mathbf{B}_{[k]v}\right] := \left[\frac{d\mathbf{x}_{[k]}}{d\mathbf{x}_{[0]}} \middle|\ \frac{d\mathbf{x}_{[k]}}{d\mathbf{v}}\right] \\ &= \frac{1}{k}\left(\mathbf{A}_{[k-1]} + \sum_{j=1}^{k-1} \mathbf{A}_{[k-j-1]x}\mathbf{B}_{[j]}\right), \qquad k = 1, \ldots, d \end{aligned} \tag{9}$$

$$\begin{aligned}\mathbf{D}_{[k]} &= \left[\mathbf{D}_{[k]x} \mid \mathbf{D}_{[k]v}\right] := \left[\frac{d\mathbf{y}_{[k]}}{d\mathbf{x}_{[0]}} \middle|\ \frac{d\mathbf{y}_{[k]}}{d\mathbf{v}}\right] \\ &= \mathbf{C}_{[k]} + \sum_{j=1}^{k} \mathbf{C}_{[k-j]x}\mathbf{B}_{[j]}, \qquad k = 0, \ldots, d \end{aligned} \tag{10}$$

Note, $\mathbf{B}_{[0]} = \left[\mathbf{I} \mid \mathbf{0}\right]$. In summary, the solutions of system (4) at $t = h$ are

$$\mathbf{x}(h) = \sum_{i=0}^{d} \mathbf{x}_{[i]}, \qquad \mathbf{y}(h) = \sum_{i=0}^{d} \mathbf{y}_{[i]} \tag{11}$$

whilst their sensitivities to initial value, $\mathbf{x}_{[0]}$ and input coefficients, $\mathbf{v}$ are

$$\mathbf{B}_x(h) := \frac{d\mathbf{x}(h)}{d\mathbf{x}_{[0]}} = \sum_{i=0}^{d} \mathbf{B}_{[i]x} = \mathbf{I} + \sum_{i=1}^{d} \mathbf{B}_{[i]x} \tag{12}$$

$$\mathbf{B}_v(h) := \frac{d\mathbf{x}(h)}{d\mathbf{v}} = \sum_{i=0}^{d} \mathbf{B}_{[i]v} = \sum_{i=1}^{d} \mathbf{B}_{[i]v} \tag{13}$$

$$\mathbf{D}_x(h) := \frac{d\mathbf{y}(h)}{d\mathbf{x}_{[0]}} = \sum_{i=0}^{d} \mathbf{D}_{[i]x} \tag{14}$$

$$\mathbf{D}_v(h) := \frac{d\mathbf{y}(h)}{d\mathbf{v}} = \sum_{i=0}^{d} \mathbf{D}_{[i]v} \tag{15}$$

4 Nonlinear Model Predictive Control

4.1 Formulation

For nonlinear system (4), at current sampling time, $t = t_0$, consider the general optimal control problem:

$$\begin{aligned} \min_{\mathbf{u}} \; & J = \psi(\mathbf{x}(t_P), \mathbf{u}(t_P)) + \int_{t_0}^{t_P} \varphi(\mathbf{x}(t), \mathbf{u}(t)) dt \\ \text{s.t. } & \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \quad \mathbf{x}(t_0) = \mathbf{x}_{[0]} \\ & \boldsymbol{\xi}(\mathbf{x}(t), \mathbf{u}(t)) \leq 0 \\ & \boldsymbol{\zeta}(\mathbf{x}(t_P), \mathbf{u}(t_P)) \leq 0 \end{aligned} \tag{16}$$

where $\boldsymbol{\xi} \in \mathbb{R}^q$ and $\boldsymbol{\zeta} \in \mathbb{R}^s$ are trajectory and terminal constraints, respectively. The prediction horizon $[t_0, t_P]$ is divided into P intervals, $t_0, t_1, \ldots, t_P$ with $t_{i+1} = t_i + h_i$ and $\sum_{i=0}^{P-1} h_i = t_P - t_0$. Assume the optimal solution to (16) is $\mathbf{u}(t) = \sum_{i=0}^{r} \mathbf{u}_{[i]}(t_k)(t - t_k)^i$ for $t_k \leq t \leq t_{k+1}$, $k = 0, \ldots, P-1$. Then, only the solution in the first interval is to be implemented and whole procedure will be repeated at next sampling instance. Note, combination of the terminal performance index ψ and the terminal constraints $\boldsymbol{\zeta}$ is imposed so that the minimized performance index in the receding sequence decreases monotonously. Hence, closed-loop stability under such moving horizon control is ensured [15].

It is well-known that the above Bolza form can be converted into the Mayer form [16]. For problem (16), augment system (4) by defining

$$\begin{aligned} \dot{\bar{x}}(t) &= \varphi(\mathbf{x}(t), \mathbf{u}(t)), \quad \bar{x}(t_0) = 0 \\ \mathbf{y}_1(t) &= \boldsymbol{\xi}(\mathbf{x}(t), \mathbf{u}(t)) \\ \mathbf{y}_2(t) &= \boldsymbol{\zeta}(\mathbf{x}(t), \mathbf{u}(t)) \\ \bar{y}(t) &= \psi(\mathbf{x}(t), \mathbf{u}(t)) + \bar{x}(t) \\ \tilde{\mathbf{x}}(t) &= \begin{bmatrix} \mathbf{x} \\ \bar{x} \end{bmatrix}, \quad \tilde{\mathbf{f}} = \begin{bmatrix} \mathbf{f} \\ \varphi \end{bmatrix}, \quad \tilde{\mathbf{x}}_{[0]} = \begin{bmatrix} \mathbf{x}_{[0]} \\ 0 \end{bmatrix} \\ \mathbf{y} &= \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \bar{y} \end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix} \boldsymbol{\xi} \\ \boldsymbol{\zeta} \\ \psi + \bar{x} \end{bmatrix} \end{aligned}$$

Then, the optimal control problem can be recast as

$$\begin{aligned} \min_{u(t)} \ & J = \bar{y}(t_P) && (17)\\ \text{s.t.} \ & \dot{\tilde{\mathbf{x}}}(t) = \tilde{\mathbf{f}}(\tilde{\mathbf{x}}(t), \mathbf{u}(t)), && \tilde{\mathbf{x}}(t_0) = \tilde{\mathbf{x}}_{[0]} \\ & \mathbf{y}(t) = \mathbf{g}(\tilde{\mathbf{x}}(t), \mathbf{u}(t)) \\ & \mathbf{y}_1(t) \le 0 && \mathbf{y}_2(t_P) \le 0 \end{aligned}$$

Let $\mathbf{u}_{[0]}(k), \ldots, \mathbf{u}_{[r]}(k)$ be input coefficients at $t = t_k$ and $\mathbf{v} \in \mathbb{R}^{m\times(r+1)\times P}$ be defined as:

$$\mathbf{v} := \begin{bmatrix} \mathbf{v}_0^T & \cdots & \mathbf{v}_{P-1}^T \end{bmatrix}^T \tag{18}$$

where $\mathbf{v}_k := \begin{bmatrix} \mathbf{u}_{[0]}^T(k) & \cdots & \mathbf{u}_{[r]}^T(k) \end{bmatrix}^T$. For given $\mathbf{v}_k$, $\tilde{\mathbf{x}}(k+1) := \tilde{\mathbf{x}}(t_{k+1})$ and $\mathbf{y}(k) := \mathbf{y}(t_k)$ are iteratively determined from $\tilde{\mathbf{x}}(k)$ using (11). Hence, (17) can be represented in discrete form

$$\begin{aligned} \min_{\mathbf{v}} \ & J = \bar{y}(P) && (19)\\ \text{s.t.} \ & \tilde{\mathbf{x}}(k+1) = \mathbf{f}_k(\tilde{\mathbf{x}}(k), \mathbf{v}_k), \quad \tilde{\mathbf{x}}(0) = \tilde{\mathbf{x}}_{[0]} \\ & \mathbf{y}(k) = \mathbf{g}_k(\tilde{\mathbf{x}}(k), \mathbf{v}_k) \quad 0 \le k \le P-1 \\ & \mathbf{y}_1(k) \le 0, \quad \mathbf{y}_2(P) \le 0 \end{aligned}$$

Problem (19) is a standard NLP problem with $P \times m \times (r+1)$ degrees of freedom. The first order derivatives of J and constraints can be easily obtained by using (14) and (15) repeatedly. More specifically, define $\frac{d\mathbf{y}(k)}{d\mathbf{v}} = \begin{bmatrix} \frac{d\mathbf{y}(k)}{d\mathbf{v}_0} & \cdots & \frac{d\mathbf{y}(k)}{d\mathbf{v}_{P-1}} \end{bmatrix}$ Then,

$$\frac{d\mathbf{y}(k)}{d\mathbf{v}_j} = \begin{cases} 0 & k \le j \\ \mathbf{D}_{\mathbf{v}}(j+1) & k = j+1 \\ \mathbf{D}_{\tilde{\mathbf{x}}}(k)\mathbf{B}_{\tilde{\mathbf{x}}}(k-1)\cdots\mathbf{B}_{\tilde{\mathbf{x}}}(j+2)\mathbf{B}_{\mathbf{v}}(j+1) & k > j+1 \end{cases}$$

Hence, derivatives of J and constraints are obtained as

$$\frac{dJ}{d\mathbf{v}} = \left[\frac{d\mathbf{y}(P)}{d\mathbf{v}}\right]_{q+s+1}, \quad \frac{d\mathbf{y}_2(P)}{d\mathbf{v}} = \left[\frac{d\mathbf{y}(P)}{d\mathbf{v}}\right]_{q+1:q+s}, \quad \frac{d\mathbf{y}_1(k)}{d\mathbf{v}_j} = \left[\frac{d\mathbf{y}(k)}{d\mathbf{v}}\right]_{1:q+1}$$

where $[\cdot]_k$ stands for the k-th row of a matrix, and $[\cdot]_{a:b}$ stands for rows of a matrix from a-th to b-th.

For MPC with moving horizon, $M < P$, *i.e.* $\mathbf{u}_k = \mathbf{u}_{M-1}$, $k = M, \ldots, P-1$, the derivative against $\mathbf{v}_{M-1}$ is a summation of derivatives against $\mathbf{v}_k$, $k = M-1, \ldots, P-1$, *i.e.* $d/d\mathbf{v}_{M-1} = \sum_{k=M-1}^{P-1} d/d\mathbf{v}_k$.

With more advanced AD programming, the second order derivatives are also

readily to be obtained [17]. Hence, using AD, the nonlinear model predictive control problem can be efficiently solved by any modern NLP software.

4.2 Error analysis and control

By using AD, the Taylor coefficients, $\mathbf{x}_{[i]}$ and $\mathbf{B}_{[i]}$ obtained using the above method are exact [11]. However, the ODE solution and sensitivity obtained at $t = h$ are only approximations due to truncation of the Taylor series. Assume $\mathbf{x}(h) = \sum_{k=0}^{d} \mathbf{x}_{[k]} h^k + \varepsilon(h, d)$ and the radius of convergence is r. Then,

$$\varepsilon(h, d) \approx C(h/r)^{d+1} \tag{20}$$

where C is constant. For a sufficiently large d,

$$r \approx r_d := \frac{\|\mathbf{x}_{[d-1]}\|_\infty}{\|\mathbf{x}_{[d]}\|_\infty} \tag{21}$$

Since $\varepsilon(h, d-1) \approx \varepsilon(h, d)(r_d/h) \approx \varepsilon(h, d) + \|\mathbf{x}_{[d]}\|_\infty$, it leads to an estimation of the truncation error:

$$\varepsilon(h, d) = \frac{h\|\mathbf{x}_{[d]}\|_\infty^2}{\|\mathbf{x}_{[d-1]}\|_\infty - h\|\mathbf{x}_{[d]}\|_\infty} \tag{22}$$

For a given error tolerance, δ, if $\delta \leq \varepsilon(h, d+1)$, either reducing h or increase d can control the error to the required level. Using (22), the required adjustment in step ($h_1 = h/c$, $c > 1$) or in order ($d_1 = d + p$, $p > 0$) to satisfy the error level can be derived:

$$c = \frac{h\|\mathbf{x}_{[d]}\|_\infty(\|\mathbf{x}_{[d]}\|_\infty + \delta)}{\|\mathbf{x}_{[d-1]}\|_\infty} \tag{23}$$

$$p = \frac{\ln(\delta/\varepsilon(h, d))}{\ln(h\|\mathbf{x}_{[d]}\|_\infty/\|\mathbf{x}_{[d-1]}\|_\infty)} \tag{24}$$

The judgement of which to be adjusted is based on the comparison of the number of operations to be increased. When reducing h by a factor of c, to reach the original step, h, the computation will increase of c times. On the other hand, if increasing d to $d + p$, computation will increase a factor of $(1 + p/d)^2$. Hence, after rounding to their nearest upper integers, if $c >= (1 + p/d)^2$, order will be increased by p. Otherwise, the step will be decreased by a factor of c.

The above error is the local error at each step. These errors will be propagated into the final cost function. The propagation can be estimated by using the

sensitivity matrix, $\mathbf{B}_x(k)$ at each step, *i.e.* the global error at step k, $\varepsilon_g(k)$ is

$$\varepsilon_g(k) = \varepsilon(h_k, d_k) + \|\mathbf{B}_x(k)\|_{i\infty}\varepsilon_g(k-1) \tag{25}$$

where $\|\cdot\|_{i\infty}$ is the induced infinity norm of a matrix. For a given process, assume $\beta \geq \|\mathbf{B}_x(k)\|_{i\infty}$, $k = 1, \ldots, P$. Then, at the end of prediction horizon, the global error is estimated as

$$\varepsilon_g(P) \leq \varepsilon(h_P, d_P) + \beta\varepsilon(h_{P-1}, d_{P-1}) + \cdots + \beta^{P-1}\varepsilon(h_1, d_1)$$

Assume all local errors are controlled at the same level, δ and the desired global error level is δ_g. Then, the local error should be controlled at level

$$\delta = \frac{\delta_g(\beta - 1)}{\beta^P - 1} \tag{26}$$

5 Case Study

5.1 Evaporator

The NMPC formulation described so far is applied to the evaporation process of Newell and Lee [18], shown in Figure 1. The process variables are listed in Table 1 and model equations are given in Appendix.

5.2 Nonlinear model predictive control

The control objective of the case study is to track setpoint changes of X_2 from 25% to 15% and P_2 from 50.5 kPa to 70 kPa when disturbances, F_1, X_1, T_1 and T_{200} are varying within ±20% of their nominal values. The control system is configured with three manipulated variables, F_2, P_{100} and F_{200} and three measurements, L_2, X_2 and P_2. All manipulated variables are subject to a first-order lag with a time constant equal to 0.5 min and saturation constraints, $0 \leq F_2 \leq 4$, $0 \leq P_{100} \leq 400$ and $0 \leq F_{200} \leq 400$. All disturbances are unmeasured and simulated as random signals changing every 5 minutes and passing through a 0.2-min first-order lag.

The NMPC is designed with cost function: $J = \int_0^P (\mathbf{y} - \mathbf{r})^T W (\mathbf{y} - \mathbf{r}) dt$, where $\mathbf{y} = \begin{bmatrix} L_2 & X_2 & P_2 \end{bmatrix}^T$ and $\mathbf{r} = \begin{bmatrix} 1 & 15 & 70 \end{bmatrix}^T$. Design parameters are: sampling period, $h = 1$ min, $P = 10$ min, input horizon $M = 5$ min and $W = \text{diag}[100, 1, 1]$. By using piecewise constant input, the result NLP problem has $3 \times M = 15$ degrees of freedom.

To fully use the advantage of the above sensitivity algorithm, the NLP problem is solved as a nonlinear least square problem [7] using the solver `lsqnonlin` in MATLAB Optimization Toolbox. To solve the problem, total $30 \times 15 = 450$ sensitivity variables have to calculated in addition to original 3 states. The sensitivity algorithm is implemented in C using ADOL-C and interfaced to MATLAB via a `mex` wrap. Simulation results with the above configuration are shown in Figure 2. It can be seen from Figure 2 that measured outputs follow the setpoints quite well (a)–(c) in spite of the existence of severe unmeasured disturbances (g)–(j). This is achieved without violating the input constraints (d)–(f).

5.3 Sensitivity algorithm comparison

To demonstrate the efficiency of the new algorithm to calculate sensitivity, the algorithm is implemented in two AD approaches: operation overloading by using ADOL-C, and source transformation, by using a preliminary AD program (STTAD) developed by the author, both in C. These two programs, both implemented with error control described in section 4.2, are compared with one of the most advanced dynamic sensitivity solvers, CVODES [5]. The comparison is based on the forward mode of CVODES, which simultaneously solves the dynamic sensitivity with the original ODE. At each step, 3 states and 18 sensitivity variables (3 states against 3 initial values and 3 input values) are integrated, and then the sensitivity of the whole prediction horizon are obtained by accumulating these stepwise sensitivity variables. All tests are done in a Windows XP PC with an Intel Pentium-4 processor running at 2.5 GHz.

Firstly, the computing times of these programs used in the above NMPC simulation are compared and shown in the first part of Table 2. It is shown that using the AD algorithm, the computation time is reduced by two orders of magnitude (from 7.08 to 0.08), whilst the ratio of the sensitivity computation time over total optimization time is reduced from over 40% to less than a percent. Hence, the original computation bottleneck does not exist when using the algorithm proposed in the work. ADOL-C is a program for general AD computation. For a specific problem, operation overloading can introduce a significant amount of computation overheads, hence reducing the efficiency. The comparison shows that for online application, source transformation is more attractive than operation overloading.

To compare computation time associated with accuracy, a reference solution is produced by using CVODES program and setting the error tolerance to the spacing of floating point number of double precision, *i.e.* $\delta = 2^{-52} = 2.2204 \times 10^{-16}$. Then, with three tolerance settings, (1e-6, 1e-8 and 1e-11), com-

putation time and accuracy of three programs are compared in the second part of Table 2. The table shows that AD programs perform better than CVODES in both efficiency and accuracy. Particularly, STTAD consistently reduces computing time about two orders of magnitude comparing with CVODES. It can be seen that the order of Taylor series plays an important role in error control. Increase the order by a few number, the error would be reduced by a number of orders of magnitude without increasing too much computation time. However, using traditional approaches, like CVODES, significant computation time may have to be traded off for a reduction in computation error.

6 Conclusion

A new algorithm to calculate non-autonomous dynamic sensitivity using AD based Taylor coefficients has been proposed. Based on the new algorithm, a NMPC formulation has been presented. Approaches for computational error analysis and control are also discussed. Due to the high-order Taylor series used, the new approach is very efficient and accurate. The feasibility of the new algorithm is demonstrated via an evaporator case study, whilst its efficiency and accuracy are verified through the comparison with CVODES, a state-of-the-art software package for solving dynamic sensitivity problems. The case study shows that the typical computation bottleneck in solving dynamic optimization problems could be removed by using the proposed dynamic sensitivity algorithm. Hence, the approach described in this work is much suitable for online application such as NMPC.

References

[1] P. B. Sistu, R. S. Gopinath, B. W. Bequette, Computational issues in nonlinear predictive control, Comput. Chem. Eng. 17 (1993) 361–367.

[2] T. Binder, L. Blank, H. Bock, R. Bulirsch, W. Dahmen, M. Diehl, T. Kronseder, W. Marquardt, J. Schloder, O. Stryk, Introduction to model based optimization of chemical processes on moving horizons, in: M. Grötschel, S. Krumke, J. Rambau (Eds.), Online Optimization of Large Scale Systems: State of Art, Springer, 2001, pp. 295–340.

[3] S.Storen, T.Hertzberg, Obtaining sensitivity information in dynamic optimization problems solved by the sequential approach, Computers and Chemical Engineering 23 (1999) 807–819.

[4] M. Schlegel, W. Marquardt, R. Ehrig, U. Nowak, Sensitivity analysis of linearly-implicit differential-algebraic systems by one-step extrapolation, Applied

Numerical Mathematics 48 (2004) 83–102.

[5] R. Serban, A. C. Hindmarch, CVODES: An ODE solver with sensitivity analysis capabilities, Tech. Rep. UCRL-JP-200039, Lawrence Livermore National Laboratory, U.S. Department of Energy (2003).

[6] R. Griesse, A. Walther, Evaluating gradients in optimal control: Continuous adjoint versus automatic differentiation, Journal of Optimization Theory and Applications 122 (1) (2004) 63–86.

[7] Y. Cao, R. Al-Seyab, Nonlinear model predictive control using automatic differentiation, in: European Control Conference (ECC 2003), Cambridge, UK, 2003, p. in CDROM.

[8] A.M.Morshedi, H.Y.Lin, R.H.Luecke, Rapid computation of the jacobian matrix for optimization of nonlinear dynamic processes, Computers and Chemical Engineering 10 (4) (1986) 367–376.

[9] L. Rall, Automatic Differentiation: Techniques and Applications, Lecture Notes in Computer Science, Vol. 120, Springer Verlag, Berlin, 1981.

[10] B. Christianson, Reverse accumulation and accurate rounding error estimates for taylor series., Optimization Methods and Software 1 (1992) 81–94.

[11] A. Griewank, Evaluating Derivatives, SIAM, Philadelphia, PA, 2000.

[12] A. Griewank, D. Juedes, J. Utke, ADOL-C: A package for the automatic differentiation of algorithms written in C/C++, ACM Transactions on Mathematical Software 22 (1996) 131–167.

[13] A. Griewank, ODE solving via automatic differentiation and rational prediction, in: D. Griffiths, G. Watson (Eds.), Numerical Analysis 1995, Vol. 344 of Pitman Research Notes in Mathematics Series, Addison-Wesley., Reading, MA, 1995.

[14] K. Röbenack, O. Vogel, Computation of state and input trajectories for flat systems using automatic differentiation, Automatica 40 (2004) 459–464.

[15] H. Chen, F. Allgöwer, A computationally attractive nonlinear predictive control scheme with guaranteed stability for stable systems, Journal of Process Control 8 (5–6) (1998) 475–485.

[16] M. Athans, P. L. Falb, Optimal Control: An Introduction to the Theory and Its Applications, McGraw-Hill, New York, 1966.

[17] B. Christianson, Cheap newton steps for optimal control problems: automatic differentiation and Pantoja's algorithm, Optimization Methods and Software 10 (5) (1999) 729–743.

[18] R. Newell, P. Lee, Applied Process Control – A Case Study, Prentice Hall, Englewood Cliffs, NJ, 1989.

Appendix. Model equations

$$\frac{\mathrm{d}L_2}{\mathrm{d}t} = \frac{F_1 - F_4 - F_2}{20} \tag{27}$$

$$\frac{\mathrm{d}X_2}{\mathrm{d}t} = \frac{F_1 X_1 - F_2 X_2}{20} \tag{28}$$

$$\frac{\mathrm{d}P_2}{\mathrm{d}t} = \frac{F_4 - F_5}{4} \tag{29}$$

$$T_2 = 0.5616 P_2 + 0.3126 X_2 + 48.43 \tag{30}$$

$$T_3 = 0.507 P_2 + 55.0 \tag{31}$$

$$F_4 = \frac{Q_{100} - 0.07 F_1 (T_2 - T_1)}{38.5} \tag{32}$$

$$T_{100} = 0.1538 P_{100} + 90.0 \tag{33}$$

$$Q_{100} = 0.16 (F_1 + F_3)(T_{100} - T_2) \tag{34}$$

$$F_{100} = Q_{100}/36.6 \tag{35}$$

$$Q_{200} = \frac{0.9576 F_{200} (T_3 - T_{200})}{0.14 F_{200} + 6.84} \tag{36}$$

$$T_{201} = T_{200} + \frac{13.68 (T_3 - T_{200})}{0.14 F_{200} + 6.84} \tag{37}$$

$$F_5 = \frac{Q_{200}}{38.5} \tag{38}$$

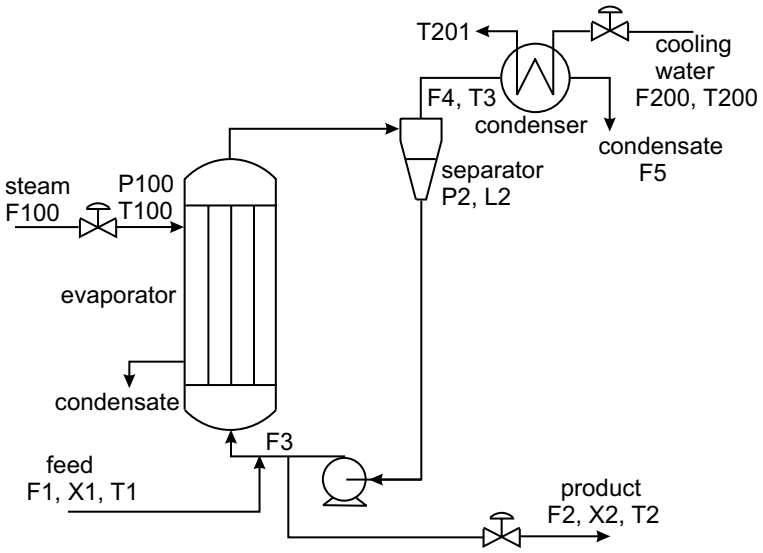

Fig. 1. Evaporator System

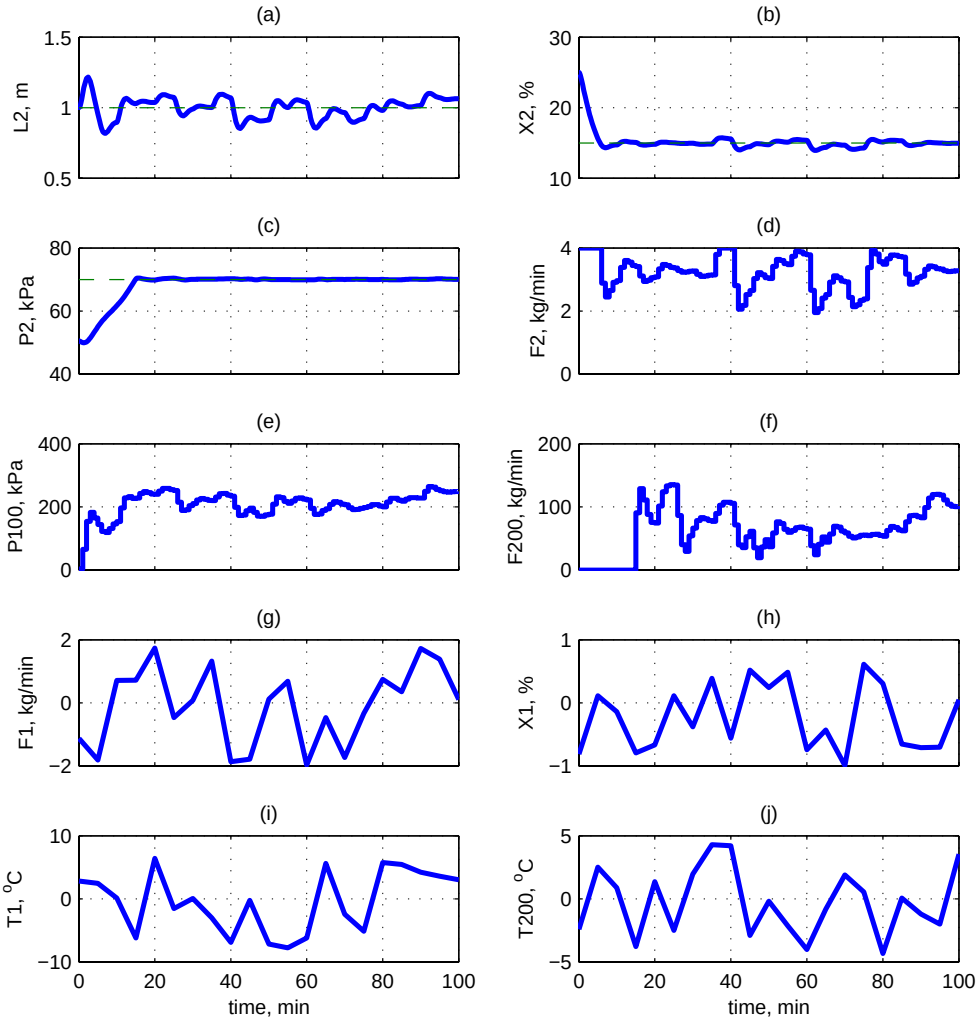

Fig. 2. Simulation result. (a)–(c) Measured outputs with setpoints. (d)–(f) Manipulated variables. (g)–(j) Disturbances.

Table 1
Variables and Optimal Values

Var.	Description	Value	Units
F_1	Feed flowrate	10	kg/mim
F_2	Product flowrate	2	kg/mim
F_3	Circulating flowrate	50	kg/mim
F_4	Vapor flowrate	8	kg/mim
F_5	Condensate flowrate	8	kg/mim
X_1	Feed composition	5	%
X_2	Product composition	25	%
T_1	Feed temperature	40	^{o}C
T_2	Product temperature	84.6	^{o}C
T_3	Vapor temperature	80.6	^{o}C
L_2	Separator level	1	meter
P_2	Operating pressure	50.5	kPa
F_{100}	Steam flowrate	9.3	kg/mim
T_{100}	Steam temperature	119.9	^{o}C
P_{100}	Steam pressure	194.7	kPa
Q_{100}	Heat duty	339	kW
F_{200}	Cooling water flowrate	208	kg/mim
T_{200}	Inlet C.W. temperature	25	^{o}C
T_{201}	Outlet C.W. temperature	46.1	^{o}C
Q_{200}	Condenser duty	307.9	kW

Table 2
Computational Time and Accuracy Comparison

NMPC							
		STTAD		ADOL-C		CVODES	
Tolerance		Time, s	T/Total, %	Time, s	T/Total, %	Time, s	T/Total, %
1e-6		0.08	0.83	2.062	17.36	7.079	42.35
Simulation, $P = 100$ and $M = 1$							
	actual	STTAD		ADOL-C		CVODES	
Tolerance	order	Time, ms	Error	Time, ms	Error	Time, ms	Error
1e-6	6	0.359	1.25e-7	7.344	1.25e-7	35.94	4.09e-5
1e-8	7	0.391	3.57e-9	8.437	3.57e-9	51.65	7.65e-7
1e-11	9	0.531	1.92e-12	11.72	1.92e-12	95.31	4.53e-9
Simulation, $P = 100$ and $M = 10$							
	actual	STTAD		ADOL-C		CVODES	
Tolerance	order	Time, ms	Error	Time, ms	Error	Time, ms	Error
1e-6	6	0.453	6.96e-8	8.125	6.96e-8	34.37	4.57e-5
1e-8	8	0.531	1.74e-9	8.750	1.74e-9	53.12	9.09e-7
1e-11	10	0.641	2.13e-13	11.72	2.13e-13	98.44	4.5326e-9
Simulation, $P = 100$ and $M = 100$							
	actual	STTAD		ADOL-C		CVODES	
Tolerance	order	Time, ms	Error	Time, ms	Error	Time, ms	Error
1e-6	6	3.281	6.96e-8	12.50	6.96e-8	42.19	4.09e-5
1e-8	8	3.281	1.74e-9	12.50	1.74e-9	59.37	7.56e-7
1e-11	10	3.437	1.85e-13	14.063	1.85e-13	107.8	4.53e-9