A FORMULATION OF NONLINEAR MODEL PREDICTIVE CONTROL USING AUTOMATIC DIFFERENTIATION

Yi Cao *,1

* *School of Engineering, Cranfield University, UK*

Abstract: The computational burden, which obstacles Nonlinear Model Predictive Control techniques to be widely adopted, is mainly associated with the requirement to solve a set of nonlinear differentiation equations and a nonlinear dynamic optimisation problem in real-time online. In this work, an efficient algorithm has been developed to alleviate the computational burden. The new approach uses the automatic differentiation techniques to solve the set of nonlinear differentiation equations, at the same time to produce the differential sensitivity of the solution against input variables. Using the differential sensitivity, the gradient of the cost function against control moves is accurately obtained so that the online nonlinear dynamic optimisation can be efficiently solved. The new algorithm has been applied to an evaporation process with satisfactory results to cope with setpoint changes, unmeasured disturbances and process-model mismatches.

Keywords: Nonlinear Control Systems, Predictive Control, Optimal Control.

1. INTRODUCTION

In last two decades, linear model predictive control has been well recognised 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. The main obstacles, which block NMPC techniques to become widely adoptable is the computational burden associated with the requirement to online solve a set of nonlinear differential equations and a nonlinear dynamic optimisation problem in real-time.

The objective of NMPC is to determine a set of future control moves (control horizon) in order to minimise a cost function based on a desired output trajectory over a prediction horizon. The introduction of nonlinear models inside the control algorithm does not cause major problems from a theoretical point of view. However, the computation involved to solving the optimisation problem at every sampling time can become so intensive, particularly for high-dimensional systems, that it could make on-line applications almost impossible (Sistu *et al.*, 1993). There exist a number of strategies for tracking the optimal control problem through nonlinear programming (NLP)(Binder *et al.*, 2001): successive linearization, direct single and multiple shooting methods, and others. In a successive linearization solution, the Jacobian linearization is performed over the prediction horizon or at a number of time steps in the prediction horizon (Ricker and Lee, 1995). Alternatively, the differential equations can be transformed into algebraic equations which are treated as nonlinear equality constraints and solved simultaneously with the NLP problem (Biegler *et al.*, 2002).In a direct single shooting approach,

[1] Email: y.cao@cranfield.ac.uk

the nonlinear dynamic optimisation problem is solved in two sequential stages, where an optimisation routine serves as an outer loop to iteratively select new sets of manipulated variable moves, while a differential equation solver is used to integrate the dynamic equations at each iteration of optimisation (Binder *et al.*, 2001).Recently, an approach using differential flatness to solve the nonlinear dynamic optimisation problem has been proposed(Mahadevan and Doyle III, 2003).

Modern NLP solvers need gradient information to perform efficiently and reliably. All approaches mentioned above requires intensive computation of derivatives. Traditionally, there are three ways to calculate sensitivity of a dynamic optimisation problem (S.Storen and T.Hertzberg, 1999): perturbations, sensitivity equations and adjoint equations. However, none of them are efficient, particularly when the number of parameters is large. Recently, automatic differentiation (AD) techniques have been applied to solve dynamic optimisation problems (Röbenack and Vogel, 2004; Griesse and Walther, 2004). In previous work, Cao and Al-Seyab (2003) used the AD techniques to approximate the solution of the sensitivity equations with efficiency significantly improved. However, only the algebraic features of AD has been utilised in the approach. Neither the efficiency nor the accuracy satisfy the requirement of NMPC.

In this work, the advantages of AD techniques have been further explored and intensively utilised to improve the efficiency of NMPC. More specifically, by calculating Taylor coefficients in forward mode and their partial derivatives in reverse mode of AD, the nonlinear differential equations are solved as a set of algebraic equations, and more importantly, the differential sensitivities can be obtained at the same time. Hence, a NMPC problem can be efficiently solved with any modern NLP software. An existing AD software package, ADOL-C (Griewank *et al.*, 1998) has been adopted with a small modification to implement the above algorithm.

The paper is organised as follows. After a brief overview of AD, its principles to solve autonomous differential equations and to calculate sensitivities are explained in section 2. Section 3 extends the techniques for non-autonomous state-space equations. Then, the formulation of NMPC using AD is proposed in section 4. 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 (Griewank, 2000). 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 subexpressions. On the other hand, with finite difference approximation, accuracy of derivatives is restricted because of cancellation and truncation errors, particularly, for higher 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 Forward and Reverse Modes

There are two computational modes of AD: the forward mode and the reverse mode. Consider a function, $y = y(v(x))$ consisting of two operations: $v = v(x)$ and $y = y(v)$. In forward mode, by applying the chain rule, $\dot{y} = dy/dx$ can be evaluated in the sequence: $\dot{x} = 1$, $\dot{v} = v'(x)\dot{x}$ and $\dot{y} = y'(v)\dot{v}$. In forward mode, a function and its derivatives can be evaluated in parallel.

The reverse mode evaluation is based on the definition of adjoint, $\overline{v} = dy/dv$. After evaluating the sequence, $v = v(x)$ and $y = y(v)$ with all intermediate results recorded, adjoints are evaluated in a reverse sequence: $\overline{y} = 1$, $\overline{v} = \overline{y}y'(v)$ and finally, $dy/dx = \overline{x} = \overline{v}v'(x)$. Evaluation in reverse mode requires more memory than in forward mode. However, when the number of dependent variables is much less than the number of independent variables, such as the objective function of an optimisation problem, evaluation in reverse mode is more efficient than in forward mode. More details of these two modes can be found elsewhere (Griewank, 2000).

2.2 Taylor Series

Consider a d-time continuously differentiable function, $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$. Let $\boldsymbol{x}(t) \in \mathbb{R}^n$ be given by the truncated Taylor series:

$$\boldsymbol{x}(t) = \boldsymbol{x}_0 + \boldsymbol{x}_1 t + \cdots + \boldsymbol{x}_d t^d \qquad (1)$$

with coefficients $\boldsymbol{x}_i = (i!)^{-1}(\partial^i \boldsymbol{x}(t)/\partial t^i)|_{t=0} \in \mathbb{R}^n$. Then, the mapped signal, $\boldsymbol{z}(t) = \boldsymbol{f}(\boldsymbol{x}(t)) \in \mathbb{R}^m$ can be expressed by a Taylor expansion:

$$\boldsymbol{z}(t) = \boldsymbol{z}_0 + \boldsymbol{z}_1 t + \cdots + \boldsymbol{z}_d t^d + \mathcal{O}(t^{d+1}) \qquad (2)$$

where $\boldsymbol{z}_j = (j!)^{-1}(\partial^j \boldsymbol{z}(t)/\partial t^j)|_{t=0} \in \mathbb{R}^m$. From the chain rule, $\boldsymbol{z}_j$ is uniquely determined by the coefficient vectors, $\boldsymbol{x}_i$ with $i \le j$, *i.e.*

$$\boldsymbol{z}_j \equiv \boldsymbol{z}_j(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_j) \tag{3}$$

Nevertheless, functions $\boldsymbol{z}_j$ are inherently d-time continuously differentiable and their derivatives satisfy the identity (Christianson, 1992):

$$\frac{\partial \boldsymbol{z}_j}{\partial \boldsymbol{x}_i} = \frac{\partial \boldsymbol{z}_{j-i}}{\partial \boldsymbol{x}_0} = \boldsymbol{A}_{j-i}(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{j-i}) \tag{4}$$

It has been proven that coefficient function, $\boldsymbol{z}_j$ is linear with respect to the upper half of its $j+1$ arguments (Griewank, 2000):

$$\begin{aligned}\boldsymbol{z}_j(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_j) = \boldsymbol{z}_j(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{k-1}, \boldsymbol{0}, \ldots, \boldsymbol{0}) \\ + \sum_{i=k}^{j} \boldsymbol{A}_{j-i}(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{j-i})\boldsymbol{x}_i\end{aligned} \tag{5}$$

for $j/2 \le k \le j$.

AD techniques provide an efficient way to solve these coefficient vectors, $\boldsymbol{z}_j$ and matrices, $\boldsymbol{A}_i$ (Griewank, 2000). For example, with the software package, ADOL-C (Griewank *et al.*, 1998), by using the forward mode of AD, all Taylor coefficient vectors for a given degree, d can be calculated simultaneously, whilst the matrices, $\boldsymbol{A}_i$ can be obtained by using the reverse mode of AD. The run time and memory requirement associated with these calculations grows only quadratically in d.

2.3 Autonomous Differential Equation

Consider an autonomous differential equation,

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t)), \qquad \boldsymbol{x}(0) = \boldsymbol{x}_0 \tag{6}$$

with d-time continuously differentiable map, $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$. Equation (6) is a special case of map $\boldsymbol{z}(t) \equiv \boldsymbol{f}(\boldsymbol{x}(t))$, where $\boldsymbol{z}(t) = \dot{\boldsymbol{x}}(t)$. Thus, for a given initial vector, $\boldsymbol{x}_0$, all Taylor coefficients, $\boldsymbol{x}_i$, $i = 1, \ldots, d$ can be iteratively determined from (3) and $\boldsymbol{x}_{i+1} = \boldsymbol{z}_i/(i+1)$. With such a series expansion up to a certain order d, solution to the initial value problem (6) can be represented as in (1) for $0 \le t \le h$ with sufficient accuracy. Thus, AD can be used to efficiently solve initial value problems (Griewank, 1995).

Moreover, by applying reverse mode of AD, a set of partial derivative matrices, $\boldsymbol{A}_{j-i} = \partial \boldsymbol{z}_j/\partial \boldsymbol{x}_i$ can also be obtained. Then, the total derivatives, $\boldsymbol{B}_k := d\boldsymbol{x}_{k+1}/d\boldsymbol{x}_0 \in \mathbb{R}^{n\times n}$ are cumulated from these matrices:

$$\begin{aligned}\boldsymbol{B}_k &= \frac{1}{k+1}\frac{d\boldsymbol{z}_k}{d\boldsymbol{x}_0} = \frac{1}{k+1}\sum_{j=0}^{k}\frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{x}_j}\frac{d\boldsymbol{x}_j}{d\boldsymbol{x}_0} \\ &= \frac{1}{k+1}\left(\boldsymbol{A}_k + \sum_{j=1}^{k}\boldsymbol{A}_{k-j}\boldsymbol{B}_{j-1}\right)\end{aligned} \tag{7}$$

Equations (3), (4) and (7) have been efficiently implemented in AD package ADOL-C (Griewank *et al.*, 1998).

3. NON-AUTONOMOUS SYSTEMS

The above algorithm is extended to solving dynamic sensitivity problems of non-autonomous state space systems:

$$\begin{aligned}\dot{\boldsymbol{x}}(t) &= \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)), & \boldsymbol{x}(0) &= \boldsymbol{x}_0 \\ \boldsymbol{y}(t) &= \boldsymbol{g}(\boldsymbol{x}(t), \boldsymbol{u}(t)), & 0 &\le t \le h\end{aligned} \tag{8}$$

where, $\boldsymbol{u}(t) \in \mathbb{R}^m$ is control input and $\boldsymbol{y}(t) \in \mathbb{R}^p$ the output. Using normalised time, $\tau = t/h$, the right-hand-side of the state equation becomes $\boldsymbol{f}_h(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau)) = h\boldsymbol{f}(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau)) = z(\tau)$ and the solution interval is $0 \le \tau \le 1$. Assume $\boldsymbol{u}(\tau) = \boldsymbol{u}_0 + \boldsymbol{u}_1\tau + \cdots + \boldsymbol{u}_r\tau^r$ and all its coefficients, $\boldsymbol{u}_j$ are known. Let $\boldsymbol{v} = \begin{bmatrix}\boldsymbol{u}_0^T & \cdots & \boldsymbol{u}_r^T\end{bmatrix}^T$. Using the forward mode of AD, the Taylor coefficients of $\boldsymbol{x}(\tau)$ and $\boldsymbol{y}(\tau)$ can be iteratively determined from $\boldsymbol{x}_0$ and $\boldsymbol{u}$.

$$\boldsymbol{x}_{j+1} = \frac{1}{j+1}\boldsymbol{z}_j(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_j, \boldsymbol{v}) \tag{9}$$

$$\begin{aligned}\boldsymbol{y}_j &= \boldsymbol{y}_j(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_j, \boldsymbol{v}) \\ &\text{for } j = 1, \ldots, d\end{aligned} \tag{10}$$

Then, by applying the reverse mode of AD, the partial derivatives are obtained and partitioned as follows:

$$\boldsymbol{A}_j = \begin{bmatrix}\boldsymbol{A}_{jx} \mid \boldsymbol{A}_{jv}\end{bmatrix} := \left[\left.\frac{\partial \boldsymbol{z}_j}{\partial \boldsymbol{x}_0}\right| \frac{\partial \boldsymbol{z}_j}{\partial \boldsymbol{v}}\right] \tag{11}$$

$$\boldsymbol{C}_j = \begin{bmatrix}\boldsymbol{C}_{jx} \mid \boldsymbol{C}_{jv}\end{bmatrix} := \left[\left.\frac{\partial \boldsymbol{y}_j}{\partial \boldsymbol{x}_0}\right| \frac{\partial \boldsymbol{y}_j}{\partial \boldsymbol{v}}\right] \tag{12}$$

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

$$\begin{aligned}\boldsymbol{B}_k &= \begin{bmatrix}\boldsymbol{B}_{kx} \mid \boldsymbol{B}_{kv}\end{bmatrix} := \left[\left.\frac{d\boldsymbol{x}_{k+1}}{d\boldsymbol{x}_0}\right| \frac{d\boldsymbol{x}_{k+1}}{d\boldsymbol{v}}\right] \\ &= \frac{1}{k+1}\left(\boldsymbol{A}_k + \sum_{j=1}^{k}\boldsymbol{A}_{(k-j)x}\boldsymbol{B}_{j-1}\right)\end{aligned} \tag{13}$$

$$\begin{aligned}\boldsymbol{D}_k &= \begin{bmatrix}\boldsymbol{D}_{kx} \mid \boldsymbol{D}_{kv}\end{bmatrix} := \left[\left.\frac{d\boldsymbol{y}_k}{d\boldsymbol{x}_0}\right| \frac{d\boldsymbol{y}_k}{d\boldsymbol{v}}\right] \\ &= \begin{bmatrix}\boldsymbol{C}_{kx} \mid \boldsymbol{0}\end{bmatrix} + \sum_{j=1}^{k}\boldsymbol{C}_{(k-j)x}\boldsymbol{B}_{j-1}\end{aligned} \tag{14}$$

In summary, the solutions of system (8) at $t = h$ are

$$\boldsymbol{x}(h) = \sum_{i=0}^{d}\boldsymbol{x}_i, \qquad \boldsymbol{y}(h) = \sum_{i=0}^{d}\boldsymbol{y}_i \tag{15}$$

whilst their sensitivities to initial value, $\boldsymbol{x}_0$ and input coefficients, $\boldsymbol{v}$ are

$$\boldsymbol{B}_x(h) := \frac{d\boldsymbol{x}(h)}{d\boldsymbol{x}_0} = \sum_{i=0}^{d}\boldsymbol{B}_{ix} \tag{16}$$

$$\boldsymbol{B}_v(h) := \frac{d\boldsymbol{x}(h)}{d\boldsymbol{v}} = \sum_{i=0}^{d} \boldsymbol{B}_{iv} \tag{17}$$

$$\boldsymbol{D}_x(h) := \frac{d\boldsymbol{y}(h)}{d\boldsymbol{x}_0} = \sum_{i=0}^{d} \boldsymbol{D}_{ix} \tag{18}$$

$$\boldsymbol{D}_v(h) := \frac{d\boldsymbol{y}(h)}{d\boldsymbol{v}} = \sum_{i=0}^{d} \boldsymbol{D}_{iv} \tag{19}$$

4. NONLINEAR MODEL PREDICTIVE CONTROL

For nonlinear system (8), at current sampling time, $t = t_0$, consider the general optimal control problem:

$$\begin{aligned} \min_{\boldsymbol{u}} J &= \psi(\boldsymbol{x}(t_P), \boldsymbol{u}(t_P)) + \int_{t_0}^{t_P} \varphi(\boldsymbol{x}(t), \boldsymbol{u}(t))dt \\ \text{s.t. } & \boldsymbol{\xi}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \le 0 \\ & \boldsymbol{\zeta}(\boldsymbol{x}(t_P), \boldsymbol{u}(t_P)) \le 0 \end{aligned} \tag{20}$$

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 (20) is $\boldsymbol{u}(t) = \sum_{i=0}^{r} \boldsymbol{u}_i(t_k)(t - t_k)^i$ for $t_k \le t \le t_{k+1}$, $k = 0, \ldots, P-1$. 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 minimised performance cost in the receding sequence decreases monotonously. Hence, closed-loop stability under such moving horizon control is ensured (Chen and Allgöwer, 1998).

To use AD techniques to solve the optimal control problem (20), it is firstly converted into the Mayer form. Augment system (8) by defining

$$\begin{aligned} \dot{x}_{n+1}(t) &= \varphi(\boldsymbol{x}(t), \boldsymbol{u}(t)), \qquad x_{n+1}(0) = 0 \\ \boldsymbol{y}_1(t) &= \boldsymbol{\xi}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \\ \boldsymbol{y}_2(t) &= \boldsymbol{\zeta}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \\ y_{q+s+1}(t) &= \psi(\boldsymbol{x}(t), \boldsymbol{u}(t)) + x_{n+1}(t) \\ \tilde{\boldsymbol{x}}(t) &= \begin{bmatrix} \boldsymbol{x} \\ x_{n+1} \end{bmatrix}, \quad \tilde{\boldsymbol{f}} = \begin{bmatrix} \boldsymbol{f} \\ \varphi \end{bmatrix}, \quad \tilde{\boldsymbol{x}}_0 = \begin{bmatrix} \boldsymbol{x}_0 \\ 0 \end{bmatrix} \\ \boldsymbol{y} &= \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ y_{q+s+1} \end{bmatrix}, \qquad \boldsymbol{g} = \begin{bmatrix} \boldsymbol{\xi} \\ \boldsymbol{\zeta} \\ \psi + x_{n+1} \end{bmatrix} \end{aligned}$$

Then, the optimal control problem can be recast as

$$\begin{aligned} \min_{u(t)} J &= y_{q+s+1}(t_P) \\ \text{s.t. } & \dot{\tilde{\boldsymbol{x}}}(t) = \tilde{\boldsymbol{f}}(\tilde{\boldsymbol{x}}(t), \boldsymbol{u}(t)), \qquad \tilde{\boldsymbol{x}}(t_0) = \tilde{\boldsymbol{x}}_0 \\ & \boldsymbol{y}(t) = \boldsymbol{g}(\tilde{\boldsymbol{x}}(t), \boldsymbol{u}(t)) \\ & \boldsymbol{y}_1(t) \le 0 \qquad\qquad\qquad \boldsymbol{y}_2(t_P) \le 0 \end{aligned} \tag{21}$$

Let $\boldsymbol{u}_0(k), \ldots, \boldsymbol{u}_r(k)$ be input coefficients at $t = t_k$ and $\boldsymbol{v} \in \mathbb{R}^{m \times r \times P}$ be defined as:

$$\boldsymbol{v} := \begin{bmatrix} \boldsymbol{v}_0^T & \cdots & \boldsymbol{v}_{P-1}^T \end{bmatrix}^T \tag{22}$$

where $\boldsymbol{v}_k := \begin{bmatrix} \boldsymbol{u}_0^T(k) & \cdots & \boldsymbol{u}_r^T(k) \end{bmatrix}^T$. For given $\boldsymbol{v}_k$, $\tilde{\boldsymbol{x}}(k+1) := \tilde{\boldsymbol{x}}(t_{k+1})$ and $\boldsymbol{y}(k) := \boldsymbol{y}(t_k)$ are iteratively determined from $\tilde{\boldsymbol{x}}(k)$ using (15). Hence, (21) can be represented in discrete form

$$\begin{aligned} \min_{\boldsymbol{v}} J &= y_{q+r+2}(P) \\ \text{s.t. } & \tilde{\boldsymbol{x}}(k+1) = \boldsymbol{f}_k(\tilde{\boldsymbol{x}}(k), \boldsymbol{v}_k), \qquad \tilde{\boldsymbol{x}}(0) = \tilde{\boldsymbol{x}}_0 \\ & \boldsymbol{y}(k) = \boldsymbol{g}_k(\tilde{\boldsymbol{x}}(k), \boldsymbol{v}_k) \qquad 0 \le k \le P-1 \\ & \boldsymbol{y}_1(k) \le 0, \qquad \boldsymbol{y}_2(P) \le 0 \end{aligned} \tag{23}$$

Problem (23) is a standard NLP problem. The first order derivatives of J and constraints can be easily obtained by using (18) and (19) repeatedly. More specifically,

$$\frac{dJ}{d\boldsymbol{v}} = \begin{bmatrix} \frac{dJ}{d\boldsymbol{v}_0} & \cdots & \frac{dJ}{d\boldsymbol{v}_{P-1}} \end{bmatrix}$$

where $dJ/d\boldsymbol{v}_{P-1} = [\boldsymbol{D}_{\boldsymbol{v}}(P)]_{q+s+1}$ (where $[\cdot]_k$ stands for the k-th row of a matrix) and for $0 \le k < P-1$,

$$\begin{aligned} \frac{dJ}{d\boldsymbol{v}_k} = ([\boldsymbol{D}_{\tilde{\boldsymbol{x}}}(P)]_{q+s+1} + [B_{\tilde{\boldsymbol{x}}}(P)]_{n+1}) \times \\ \boldsymbol{B}_{\tilde{\boldsymbol{x}}}(P-1) \cdots \boldsymbol{B}_{\tilde{\boldsymbol{x}}}(k+2) \boldsymbol{B}_{\boldsymbol{v}}(k+1) \end{aligned}$$

Similarly,

$$\begin{aligned} \frac{d\boldsymbol{y}_2(P)}{d\boldsymbol{v}} &= \begin{bmatrix} \frac{d\boldsymbol{y}_2(P)}{d\boldsymbol{v}_0} & \cdots & \frac{d\boldsymbol{y}_2(P)}{d\boldsymbol{v}_{P-1}} \end{bmatrix} \\ \frac{d\boldsymbol{y}_2(P)}{d\boldsymbol{v}_{P-1}} &= [\boldsymbol{D}_{\boldsymbol{v}}(P)]_{q+1:q+s} \\ \frac{d\boldsymbol{y}_2(P)}{d\boldsymbol{v}_k} &= [\boldsymbol{D}_{\tilde{\boldsymbol{x}}}(P)]_{q+1:q+s} \times \\ & \qquad \boldsymbol{B}_{\tilde{\boldsymbol{x}}}(P-1) \cdots \boldsymbol{B}_{\tilde{\boldsymbol{x}}}(k+2) \boldsymbol{B}_{\boldsymbol{v}}(k+1) \end{aligned}$$

Finally,

$$\frac{d\boldsymbol{y}_1(k)}{d\boldsymbol{v}_j} = \begin{cases} 0 & k \le j \\ [D_{\boldsymbol{v}}(j+1)]_{1:q} & k = j+1 \\ [D_{\tilde{\boldsymbol{x}}}(k)]_{1:q} \boldsymbol{B}_{\tilde{\boldsymbol{x}}}(k-1) \cdots & \\ \boldsymbol{B}_{\tilde{\boldsymbol{x}}}(j+2) \boldsymbol{B}_{\boldsymbol{v}}(j+1) & k \ge j+1 \end{cases}$$

With more advanced AD programming, the second order derivatives are also readily to be obtained (Christianson, 1999). Hence, using AD, the nonlinear model predictive control problem can be efficiently solved by using any modern NLP software.

5. CASE STUDY

5.1 Evaporator

The NMPC formulation described so far is applied to the evaporation process of Newell and Lee (1989), shown in Figure 1. This is a "forced-circulation" evaporator, where the concentration

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

of dilute liquor is increased by evaporating solvent from the feed stream through a vertical heat exchanger with circulated liquor. The process variables are listed in Table 1 and model equations are given in Appendix A.

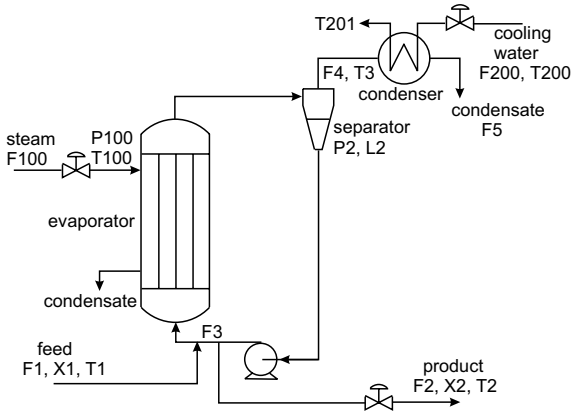

Fig. 1. Evaporator System

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 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: $J = \int_0^P (\boldsymbol{y} - \boldsymbol{r})^T W (\boldsymbol{y} - \boldsymbol{r}) dt$. Design parameters are:

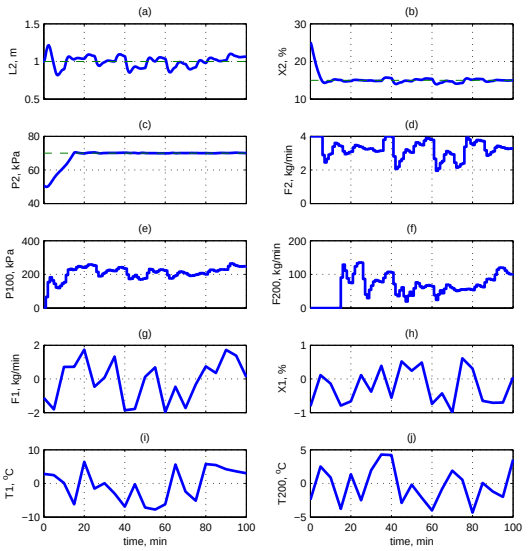

Fig. 2. Simulation result. (a)–(c) Measured outputs with setpoints. (d)–(f) Manipulated variables. (g)–(j) Disturbances.

sampling period, $h = 1$ min, $P = 5$ min, input horizon, $M = 2$ min, $W \equiv \text{diag}[100, 1, 1]$. The nonlinear dynamic model of the process is used as the prediction model for NMPC whilst the actuator lags and disturbances lags are ignored from the prediction. The ignored lags play as process-model mismatches for the NMPC.

Simulation is performed with the above configuration. The results 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). Therefore, the NMPC controller is effective and satisfies the performance requirements proposed.

To demonstrate the efficiency of the new formulation (C1), for Taylor expansion order, $d = 2$ and $d = 5$, the computational times are compared with other three NMPC controllers (Table 2): C2 using MATLAB ODE23 solver plus perturbation to get sensitivity, C3 using ODE23 plus approximated sensitivity using AD (Cao and Al-Seyab, 2003) and C4 using AD to solve the differential equations but with perturbation to get sensitivity. Table 2 shows that the differential equation solver using AD reduces computational time by an order of magnitude (comparing C1 with C3 and C2 with C4) and sensitivity calculation using AD saves another order of magnitude in time (comparing C1 with C4 and C2 with C3). For differentiation using perturbation approaches, the computational time is very sensitive to the number of independent variables (C2 and C4), whilst for AD approaches, it is insensitive in this case study (C1 and C3).

Table 2. Computational time (seconds)

M	C1		C2	C3	C4	
	$d=2$	$d=5$			$d=2$	$d=5$
2	0.732	0.915	46.05	9.50	4.14	5.00
3	0.628	0.826	61.12	8.42	5.00	5.16
4	0.546	0.673	63.81	8.05	5.56	7.33
5	0.560	0.691	70.20	8.35	6.89	8.55

6. CONCLUSION

A new NMPC formulation using AD has been proposed. The new algorithm reduces computational time both in differential equation solving and sensitivity calculations. Hence, it significantly improves the efficiency of NMPC. This approach can be extended to other nonlinear problems, such as nonlinear model identification and nonlinear state estimation.

REFERENCES

Biegler, L.T., A.M. Cervantes and A. Wächter (2002). Advances in simultaneous strategies for dynamic process optimization. *Chemical Engineering Science* **57**, 575–593.

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

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

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

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

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

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

Griewank, A. (1995). Ode solving via automatic differentiation and rational prediction. In: *Numerical Analysis 1995* (D.F. GriPths and G.A. Watson, Eds.). Vol. 344 of *Pitman Research Notes in Mathematics Series*. Addison-Wesley.. Reading, MA.

Griewank, A. (2000). *Evaluating Derivatives*. SIAM. Philadelphia, PA.

Griewank, Andreas, David Juedes3, Hristo Mitev, Jean Utke, Olaf Vogel and Andrea Walther (1998). *ADOL-C: A Package for the Automatic Differentiation of Algorithms Written in C/C++*. version 1.8 ed.

Mahadevan, R. and F. J. Doyle III (2003). Efficient optimization approaches to nonlinear model predictive control. *International Journal of Robust and Nonlinear Control* **13**, 309–329.

Newell, R.B. and P.L. Lee (1989). *Applied Process Control – A Case Study*. Prentice Hall. Englewood Cliffs, NJ.

Ricker, N.L. and J.H. Lee (1995). Nonlinear model predictive control of the tennessee eastman challenge process. *Computers and Chemical Engineering* **19**(9), 961–981.

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

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

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

Appendix A. MODEL EQUATIONS

$$\frac{\mathrm{d}L_2}{\mathrm{d}t} = \frac{F_1 - F_4 - F_2}{20} \tag{A.1}$$

$$\frac{\mathrm{d}X_2}{\mathrm{d}t} = \frac{F_1 X_1 - F_2 X_2}{20} \tag{A.2}$$

$$\frac{\mathrm{d}P_2}{\mathrm{d}t} = \frac{F_4 - F_5}{4} \tag{A.3}$$

$$T_2 = 0.5616 P_2 + 0.3126 X_2 + 48.43 \tag{A.4}$$

$$T_3 = 0.507 P_2 + 55.0 \tag{A.5}$$

$$F_4 = \frac{Q_{100} - 0.07 F_1 (T_2 - T_1)}{38.5} \tag{A.6}$$

$$T_{100} = 0.1538 P_{100} + 90.0 \tag{A.7}$$

$$Q_{100} = 0.16 (F_1 + F_3)(T_{100} - T_2) \tag{A.8}$$

$$F_{100} = Q_{100}/36.6 \tag{A.9}$$

$$Q_{200} = \frac{0.9576 F_{200} (T_3 - T_{200})}{0.14 F_{200} + 6.84} \tag{A.10}$$

$$T_{201} = T_{200} + \frac{13.68 (T_3 - T_{200})}{0.14 F_{200} + 6.84} \tag{A.11}$$

$$F_5 = \frac{Q_{200}}{38.5} \tag{A.12}$$