On Gradient Computation in Single-shooting Nonlinear Model Predictive Control

Ruben Ringset * **Lars Imsland** *,** **Bjarne Foss** **

* *Cybernetica AS, Leirfossveien 27, N-7038 Trondheim, Norway. (e-mail: ruben.ringset@cybernetica.no).*
** *Department of Engineering Cybernetics, NTNU, N-7038, Trondheim, Norway. (e-mail: lars.imsland,bjarne.foss@itk.ntnu.no)*

Abstract: This paper gives an overview of methods for computing derivative information in dynamic optimization with path constraints. Efficiency of forward and adjoint techniques are discussed in a discrete-time setting and some algorithms are derived. Next, the discussion is extended to also include continuous-discrete systems. Dimensions in the model, signal parameterization, horizon length and sampling interval affect each of the methods differently. The key contributions of this paper is to give an overview of these methods, how they can be combined, and how different parameters affect efficiency.

Keywords: dynamic optimization, nonlinear model predictive control, adjoint method, sensitivity analysis, sequential quadratic programming.

1. INTRODUCTION

Algorithms for optimization-based control using nonlinear dynamic models such as nonlinear model predictive control (NMPC) formulates an 'open-loop' constrained dynamic optimization problem, which is re-solved and re-implemented at regular sampling intervals. This combines the advantage of the numerical optimal control solution with the feedback achieved through updated model information (measurements, and estimated states and parameters).

The NMPC nonlinear dynamic optimization problem is typically discretized and solved by applying quadratic programming sequentially to quadratic/linear approximations of the optimization problem (*Sequentially Quadratic Programming*, SQP). SQP algorithms for NMPC may be categorized based on how they discretize the dynamic optimization problem. Broadly speaking, algorithms using only manipulated variables as optimization variables are often called *single shooting* algorithms (or *sequential*, as numerical optimization is executed sequential to numerical simulation for gradient computation), while algorithms using in addition a discretization of the future process model variables as optimization variables, are often called *simultaneous* algorithms. In-between solutions, offering a trade-off between the traits of the two extremes, are often called *multiple shooting*. Such algorithms are discussed in e.g. Li and Biegler (1989); Biegler et al. (2002); Bock et al. (2000).

This paper is concerned with gradient computation for single-shooting NMPC optimization, using a setup similar to Li and Biegler (1989); De Oliveira and Biegler (1995). Our aim is to give a clearer picture of the possibilities and trade-offs involved in the computation of the gradient, in particular the impulse- (or step-) response matrix. First, we use a pure discrete-time setting to focus on the algorithms for constructing the impulse-response matrices. In particular we demonstrate how using an adjoint-based approach amounts to constructing the impulse-response matrices row-wise, rather than column-wise as a 'forward' implementation would do. Some numerical examples illustrate that for 'fat' systems this can be advantageous.

Thereafter, we complement the picture by evaluating alternatives when the discrete-time model originates from simulating a continuous-time model.

2. SINGLE-SHOOTING DISCRETE-TIME DYNAMIC OPTIMIZATION

Assume that the nonlinear system model is on the form

$$x_{k+1} = f(x_k, u_k), \tag{1a}$$
$$z_k = g(x_k, u_{k-1}), \tag{1b}$$

where f and g are differentiable functions. For simplicity assume that

$$f(0,0) = 0, \quad g(0,0) = 0, \tag{2}$$

and that the origin is the desired settling point for both the states and the control inputs. Note, however, that the results presented throughout this paper can be generalized to, and is probably most useful for, time-varying reference signals.

Under the above assumptions, the dynamic optimization problem to be solved in an NMPC implementation, is stated as

$$\min J = \frac{1}{2}\sum_{k=0}^{N-1}\left(z_{k+1}^T Q z_{k+1} + u_k^T R u_k\right), \tag{3a}$$

$$\text{s.t. } x_0 = x(t_0), \tag{3b}$$

$$x_{k+1} = f(x_k, u_k),\ k \in \{0,\dots,N-1\}, \tag{3c}$$

$$z_k = g(x_k, u_{k-1}),\ k \in \{1,\dots,N\}, \tag{3d}$$

$$u_{\min} \le u_k \le u_{\max},\ k \in \{0,\dots,N-1\}, \tag{3e}$$

$$z_{\min} \le z_k \le z_{\max},\ k \in k \in \{1,\dots,N\}, \tag{3f}$$

in which $Q = Q^T \ge 0$, $R = R^T > 0$, $x_k \in \mathbb{R}^{n_x}$ are the state variables, $u_k \in \mathbb{R}^{n_u}$ the controlled inputs and $z_k \in \mathbb{R}^{n_z}$ the controlled outputs. Since x_0 is fixed, $\{x_k\}_{k=1}^N$ and $\{z_k\}_{k=1}^N$ are uniquely determined by $\{u_k\}_{k=0}^{N-1}$. Using a single shooting formulation the optimization problem is posed regarding $\{x_k\}_{k=1}^N$ and $\{z_k\}_{k=1}^N$ as implicit functions of $\{u_k\}_{k=0}^{N-1}$, which are obtained by simulation, thus removing the model equations as equality constraint in (3).

Define

$$u = \begin{pmatrix} u_0 \\ \vdots \\ u_{N-1} \end{pmatrix}, \quad \bar{R} = \begin{pmatrix} R & & 0 \\ & \ddots & \\ 0 & & R \end{pmatrix}, \tag{4}$$

$$z = \begin{pmatrix} z_1 \\ \vdots \\ z_N \end{pmatrix}, \quad \bar{Q} = \begin{pmatrix} Q & & 0 \\ & \ddots & \\ 0 & & Q \end{pmatrix}, \tag{5}$$

and let the impulse response matrix of the linearized system model be defined by

$$\Xi = \begin{pmatrix} C_1B_0 + D_1 & 0 & 0 & \cdots \\ C_2A_1B_0 & C_2B_1 + D_2 & 0 & \cdots \\ C_3A_2A_1B_0 & C_3A_2B_1 & C_3B_2 + D_3 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}, \tag{6}$$

where

$$A_k = \frac{\partial f(x_k, u_k)}{\partial x_k}, \qquad B_k = \frac{\partial f(x_k, u_k)}{\partial u_k}, \tag{7}$$

$$C_k = \frac{\partial g(x_k, u_{k-1})}{\partial x_k}, \quad D_k = \frac{\partial g(x_k, u_{k-1})}{\partial u_{k-1}}. \tag{8}$$

Note that the impulse response matrix includes the sensitivities from every controlled input u_k to each of the outputs z_k on the horizon, i.e.

$$\Xi[i,j] = \frac{\partial z_i}{\partial u_{j-1}} \tag{9}$$

Thus, the gradient and, perhaps remarkably, also the Hessian of J are given by

$$\nabla_u J = \Xi \bar{Q} z + \bar{R} u, \tag{10}$$

$$\nabla_{uu}^2 J = \Xi^T \bar{Q} \Xi + \bar{R}, \tag{11}$$

which can be used to compute the Newton search direction,

$$p = -[\nabla_{uu}^2 J]^{-1}[\nabla_u J], \tag{12}$$

in for instance an SQP algorithm.

The constraint gradient for u is trivial, while the constraint gradient for z is given from Ξ. Thus, all the required derivative information is given from Ξ.

In (3) the inputs u_k are used directly as free optimization variables, while it in many cases is desirable to use the input moves $\Delta u_k = u_k - u_{k-1}$ instead. In this case, the impulse response matrix is more-or-less replaced by the step response matrix, which can easily be calculated from the impulse response matrix. Most observations in this paper holds for this case as well.

The concept of *input blocking* (Maciejowski, 2001) (or similar blocking concepts (Cagienard et al., 2007)) is very important to reduce computational complexity. Input blocking consists of dividing (individually or collectively) the input horizon into blocks where the inputs are kept constant. To keep the discussion simple, we avoid introducing input blocking at this stage, but we will comment on it later on.

In situations where the output constraints $z_{\min} < z_k < z_{\max}$ are not present $\nabla_u J$ can be calculated by a simpler and much more computationally efficient method than by first obtaining Ξ. It is well known that adjoint methods are very efficient for obtaining sensitivities of a low dimensional function with respect to a large number of parameters. Using the adjoint method, calculating $\nabla_u J$ only requires one simulation in forward time and one in reverse time. See e.g. Cao et al. (2003) which presents adjoint methods for obtaining sensitivities for parameter-dependent differential algebraic equation systems (DAEs) up to index two, and Jørgensen (2007) who discusses calculation of derivative information by adjoint methods in the context of nonlinear model predictive control (NMPC), nonlinear moving horizon estimation (MHE) and continuous-discrete system models.

3. CALCULATION OF IMPULSE RESPONSE MATRIX

As noted above, all the necessary derivative information required in each SQP iteration of solving (3) is given by Ξ. In this section we will elaborate on different methods and choices in order to calculate the impulse response matrix. Since gradient computation is the most computationally expensive part of a typical NMPC algorithm, finding efficient algorithms is important as a means to increase overall efficiency.

In this section the focus is on a discrete-time setting. For discrete-time systems originating from simulation of continuous-time systems, this can be interpreted as a focus on the computational cost in gradient computation except for simulation. Later, the discussion will be extended to also include simulation (that is, considering continuous-discrete models).

3.1 Finite Differences

In situations where the linearized system matrices A_k, B_k, C_k, and D_k are unavailable Ξ may be found by using finite difference techniques. Using one-sided finite differences requires a number of $n_u \cdot N + 1$ simulations. In complex applications, choosing the right size for the perturbation parameter can be a delicate matter since there is a trade-off between loss of accuracy due to final precision arithmetic and error in approximation of the exact derivative by not choosing an infinitesimally small perturbation value.

3.2 Forward Method

The following algorithm constructs the impulse response matrix by the forward method. Structure is exploited in a

column-wise manner by noting that adjacent elements in each column only differ by two matrix multiplications from left. By inspecting the progress of the algorithm the reader may observe that the matrix is constructed in forward time, meaning that A_k is inserted before A_{k+1}.

Algorithm 1 Ξ - Forward method

for $j = 1$ to N **do**
  for $i = 1$ to $j-1$ **do**
    $\psi[1,i] = A_{j-1}\psi[1,i]$
  end for
  $\psi[1,j] = B_{j-1}$
  for $i = 1$ to $j-1$ **do**
    $\Xi[j,i] = C_j\psi[1,i]$
  end for
  $\Xi[j,j] = C_j\psi[1,j] + D_j$
end for

3.3 Adjoint Method

As opposed to forward methods, adjoint methods compute derivative information by applying the chain rule in a reverse manner. Forward methods are generally best suited for obtaining sensitivity of a potentially high dimensional function or functional with respect to a small number of parameters. Adjoint methods becomes efficient for the opposite case, where the sensitivity of a low dimensional function or functional with respect to a large number of parameters is desired. In the following we derive the adjoint equations for calculating the impulse response matrix.

Define the function

$$\mathcal{L}_j = z_j - \sum_{i=0}^{j-1} \left[\lambda_{i+1}^T (x_{i+1} - f(x_i, u_i))\right]. \tag{13}$$

In order to comply with the nonlinear model, the first variation of $\mathcal{L}_j$ is identical to the first variation of z_j given by

$$\begin{aligned}\delta z_j = &\left(\frac{\partial z_j}{\partial x_j}\delta x_j + \frac{\partial z_j}{\partial u_{j-1}}\delta u_{j-1}\right) \\ &+ \sum_{i=0}^{j-1}\left(\lambda_{i+1}^T \frac{\partial f(x_i,u_i)}{\partial u_i}\delta u_i\right) \\ &+ \sum_{i=0}^{j-1}\left(-\lambda_{i+1}^T\delta x_{i+1} + \lambda_{i+1}^T\frac{\partial f(x_i,u_i)}{\partial x_i}\delta x_i\right).\end{aligned} \tag{14}$$

By inserting the identity

$$\sum_{i=0}^{j-1}\lambda_{i+1}^T\delta x_{i+1} = \sum_{i=0}^{j-1}\lambda_i^T\delta x_i + \lambda_j^T\delta x_j - \lambda_0^T\delta x_0, \tag{15}$$

substituting the Jacobian matrices and rearranging, we obtain

$$\begin{aligned}\delta z_j = C_j\delta x_j + \sum_{i=0}^{j-1}\left(\left[\lambda_{i+1}^T A_i - \lambda_i^T\right]\delta x_i\right) - \lambda_j^T\delta x_j \\ + \lambda_0^T\delta x_0 + D_j\delta u_{j-1} + \sum_{i=0}^{j-1}\left(\lambda_{i+1}^T B_i\delta u_i\right).\end{aligned} \tag{16}$$

By noting that $\delta x_0 = 0$ and letting λ_i fulfill

$$\lambda_j^T = C_j, \tag{17a}$$

$$\lambda_i^T = \lambda_{i+1}^T A_i,\ i \in \{j-1,\dots,1\}, \tag{17b}$$

the δx_i terms vanish and the gradient is given by

$$\frac{\partial z_j}{\partial u_i} = \frac{\delta z_j}{\delta u_i} = \begin{cases}\lambda_j^T B_{j-1} + D_j, & i = j-1 \\ \lambda_{i+1}^T B_i, & i \in \{j-2,\dots,0\}\end{cases}. \tag{18}$$

Calculation of the impulse response matrix by the adjoint method is done in Algorithm 2 which obtains each of the elements in Ξ by iterating (17) and (18) backwards.

Algorithm 2 Ξ - Adjoint method

for $k = N$ to 1 **do**
  $\lambda[k,1] = C_k$
  for $i = k+1$ to N **do**
    $\lambda[i,1] = \lambda[i,1]A_k$
  end for
  $\Xi[k,k] = \lambda[k,1]B_{k-1} + D_k$
  for $i = k+1$ to N **do**
    $\Xi[i,k] = \lambda[i,1]B_{k-1}$
  end for
end for

An unavoidable drawback with adjoint methods is that in order to reconstruct the information needed in the reverse solve, the state trajectory $\{x_k\}_{k=1}^N$ must be stored during the forward solve. Also note that if A_k and B_k cannot be reconstructed directly from x_k and u_k they will also need to be stored in memory. This will for instance be the case for continuous-discrete models where A_k and B_k are obtained by integration of sensitivity equations. Continuous-discrete models and integration of sensitivities are discussed in more detail later.

3.4 Comparison of forward and adjoint method

The difference between Algorithm 1 (forward method) and Algorithm 2 (adjoint method) is that Algorithm 1 exploits that adjacent elements in each column of Ξ only differ by two matrix multiplications from left while Algorithm 2 exploits that adjacent elements in each row of Ξ only differ by two matrix multiplications from right. Hence, the adjoint method exploits structure row-wise by constructing Ξ in reverse time while the forward method exploits structure column-wise by constructing Ξ in forward time.

Since the order of the matrix multiplications in Algorithm 1 and Algorithm 2 are not the same, they will also differ in the total number of single multiplications. The total number of multiplications is

$$\begin{aligned}n_x^2 m\sum_{k=1}^{N-1}k + n_z n_x n_u\sum_{k=1}^{N}k = \\ \frac{n_x^2 mN(N-1) + n_z n_x n_u N(N+1)}{2},\end{aligned} \tag{19}$$

where $m = n_u$ for Algorithm 1 and $m = n_z$ for Algorithm 2.

Therefore, we conclude that the adjoint method is best suited for problems where $n_z < n_u$, while the opposite is true if $n_u < n_z$. If $n_z = n_u$ and the full impulse response matrix is to be found, the forward method should be preferred as generation of Ξ may be done in parallel with the simulation opposed to the adjoint method where the entire state trajectory must be obtained and stored in memory first.

In applications one may consider using a different parameterization of the output trajectory than in (3). If only certain 'coincidence points' of the controlled variables z_k on the prediction horizon of length N are included in the optimization criterion J as well as in the output constraints $z_{\min} < z_k < z_{\max}$ only the corresponding rows of Ξ will be required in order to construct derivative information used in the optimization algorithm. By choosing a parameterization of the controlled variables with $N_z < N$ coincidence points, Algorithm 2 may easily be modified such that only the corresponding rows are calculated. Also note that due to the lower block triangular structure of Ξ, removing output variables close to the end of the horizon will have greater impact on computation time compared to removing them early in the horizon. Note that Algorithm 1 needs to generate the full impulse response matrix even though only some of the rows are needed.

It is tempting to believe that applying input blocking to the control variables u_k will result in a similar reduction in computation complexity for the forward method by making some of the columns of Ξ redundant, but unfortunately this is not the case. Even though some of the control variables are not allowed to change at certain sample instants, their sensitivity still needs to be included in the optimization. That is, using input blocking will not reduce complexity in *discrete-time* gradient computation unless finite differences are used.

The following figures show runtime as function of horizon length when computing Ξ using Algorithm 1 and Algorithm 2. Runtime for a modified version of Algorithm 2 is also shown. The modified version of Algorithm 2 assumes a parameterization where the output variables are included in the optimization every fifth sample. Thus only every fifth row of Ξ is calculated.

In Figure 1 $n_u < n_z$, and thus as expected the forward method is best suited. The simulated runtimes for Algorithm 1 relative to the runtime for Algorithm 2 was found to be in very good accordance with the total number of multiplications in each of the algorithms given by (19).

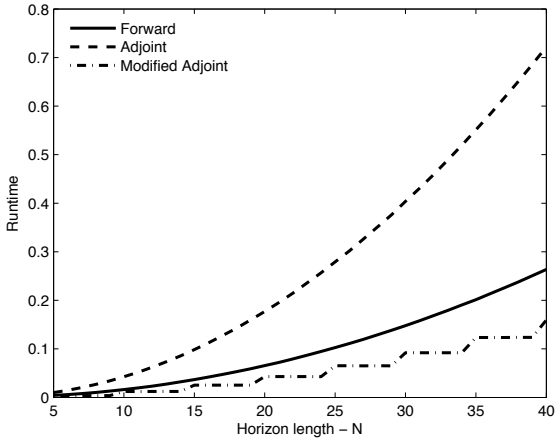

Fig. 1. Runtime - Calculate Ξ ($n_x = 40$, $n_u = 5$, $n_z = 20$)

Results for the opposite situation, i.e. $n_z < n_u$, are shown in Figure 2. This time, again as expected, Figure 2 illustrates that Algorithm 2 is faster than Algorithm 1.

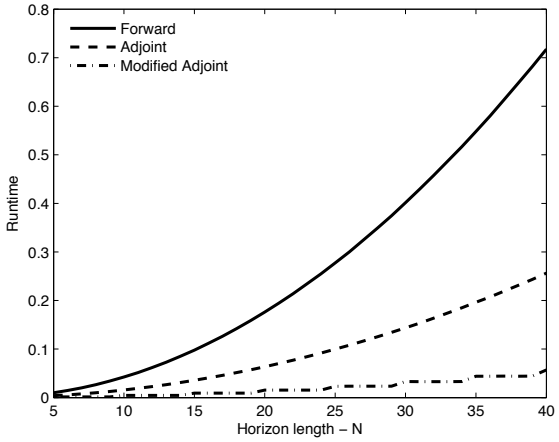

Fig. 2. Runtime - Calculate Ξ ($n_x = 40$, $n_u = 20$, $n_z = 5$)

Both figures also show runtime for a modified version of Algorithm 2. The modified adjoint method assumes a parameterization where the output variables are included in the optimization every fifth sample. As noted earlier, the adjoint method can benefit from this, as calculation of some of the rows in Ξ can be omitted. Comparison of the figures demonstrate that runtime for the different algorithms is highly dependent on the dimensions in the model, and the parameterization of z_k.

It should be noted that in generation of Figures 1 and 2 we have implemented matrix multiplications 'by hand' to avoid overhead and optimization/parallelization that might appear in use of a matrix library.

4. CONTINUOUS-DISCRETE SYSTEMS

In this section the discussion will be extended to include continuous-discrete models. Assume that the system model is on ODE form,

$$\dot{x}(t) = \phi(x(t), u(t)), \tag{20a}$$

$$z(t) = \gamma(x(t), u(t)). \tag{20b}$$

Moreover, assume that the control signal is parameterized using zero order hold, i.e.,

$$u(t) = u(kT_s) = u_k, \quad kT_s \leq t < (k+1)T_s. \tag{21}$$

4.1 Sensitivity method 1

To handle models on this form in the previous discrete-time framework, one should use ODE solvers/integrators to obtain a discrete-time formulation (1). At the same time, one should calculate the discrete-time Jacobians A_k and B_k. While using finite differences, or direct differentiation in case of fixed-step ODE solvers, is possible, it is in general preferable to use *sensitivity integration* (e.g. Hairer et al. (1993)).

By defining

$$W_k(t) = \frac{\partial x(t)}{\partial x(kT_s)}, \quad S_k(t) = \frac{\partial x(t)}{\partial u(kT_s)}, \tag{22}$$

the discrete-time Jacobians are given by

$$A_k = \frac{\partial x([k+1]T_s)}{\partial x(kT_s)} = W_k([k+1]T_s), \tag{23}$$

$$B_k = \frac{\partial x([k+1]T_s)}{\partial u(kT_s)} = S_k([k+1]T_s). \tag{24}$$

The matrices W_k and S_k may be obtained by integration of the sensitivity equations

$$\dot{S}_k(t) = \frac{\partial \phi}{\partial x}(t)S_k(t) + \frac{\partial \phi}{\partial u}(t), \tag{25a}$$

$$\dot{W}_k(t) = \frac{\partial \phi}{\partial x}(t)W_k(t). \tag{25b}$$

where the continuous-time Jacobians are found either symbolically, using automatic differentiation, or by finite differences, in order of decreasing overall efficiency. The sensitivity ODEs (25) are solved for each sampling interval with initial values reset to

$$W_k(kT_s) = I, \quad S_k(kT_s) = 0, \tag{26}$$

at every sampling instant. This approach is incorporated in Algorithm 3 which may be combined with either Algorithm 1 or Algorithm 2 to construct Ξ. While the above is well known in the literature (e.g. Li and Biegler (1989)), the purpose of including them here is to investigate how they may be combined with Algorithm 1 and Algorithm 2 and to elaborate on some practical issues that may be of importance for achieving an efficient implementation. Note that combining Algorithm 3 and 2 requires that the sensitivities A_k, B_k are stored during the forward solve as they are needed in reverse order when constructing Ξ. Hence, this method may easily be constrained by limited internal memory size.

Algorithm 3 Sensitivity integration - method 1

$x_0 = x_{init}$
for $k = 0$ to $N-1$ **do**
  $S_k(kT_s) = 0$
  $W_k(kT_s) = I$
  Integrate from $t = kT_s$ **to** $t = (k+1)T_s$
    $\dot{x}(t) = \phi(x(t), u(t))$
    $\dot{S}_k(t) = \frac{\partial \phi}{\partial x}(t)S_k(t) + \frac{\partial \phi}{\partial u}(t)$
    $\dot{W}_k(t) = \frac{\partial \phi}{\partial x}(t)W_k(t)$
end for

Even though the dimension of the matrix ODEs (25) may be large, the fact that they share Jacobians with (20) may be exploited in specialized algorithms for sensitivity integration (Hindmarsh and Serban, 2006; Schlegel et al., 2004). However, as the dimension of the sensitivity ODE (25) is $n_x^2 + n_x n_u$, integrating (25) for system models with a large number of states, may become a daunting task, even if structure is exploited.

4.2 Sensitivity method 2

For system models where $n_x \gg n_u$ one might instead consider using an alternative method for obtaining the sensitivities (Li and Biegler, 1989).

By defining the ODE system

$$\dot{S}_k(t) = \begin{cases} \frac{\partial \phi}{\partial x}(t)S_k(t) + \frac{\partial \phi}{\partial u}(t), & kT_s \leq t < (k+1)T_s \\ \frac{\partial \phi}{\partial x}(t)S_k(t), & t \geq (k+1)T_s \end{cases} \tag{27}$$

all the sensitivities required to construct Ξ may be obtained without having to integrate $W_k(t) \in \mathbb{R}^{n_x \times n_x}$. By using (27) instead of (25) the dimension of the sensitivity ODE is reduced from $n_x(n_x+n_u)$ to $n_x n_u$, but this smaller ODE must be simulated several times. That is, the total integration length is increased as can be seen by inspecting Algorithm 4.

Algorithm 4 Sensitivity integration - method 2

$x_0 = x_{init}$
for $k = 0$ to $N-1$ **do**
  Integrate from $t = kT_s$ **to** $t = (k+1)T_s$
    $\dot{x}(t) = \phi(x(t), u_k)$
    $\dot{S}_k(t) = \frac{\partial \phi}{\partial x}(t)S_k(t) + \frac{\partial \phi}{\partial x}(t)$
  for $n = k+1$ to $N-1$ **do**
    Integrate from $t = nT_s$ **to** $t = (n+1)T_s$
      $\dot{x}(t) = \phi(x(t), u_n)$
      $\dot{S}_k(t) = \frac{\partial \phi}{\partial x}(t)S_k(t)$
  end for
end for

Using Algorithm 4, the sensitivities $S_k(t)$ cannot be directly used in Algorithm 1 and 2, since the sensitivities $W_k(t)$ have not been calculated. Instead, the following simple algorithm can be combined with Algorithm 4 in order to calculate Ξ.

Algorithm 5 Sensitivity integration - method 2

for $k = 1$ to N **do**
  $\Xi[k,k] = C_k B_{k-1} + D_k$
  for $n = k+1$ to N **do**
    $\Xi[n,k] = C_n S_{k-1}(nT_s)$
  end for
end for

4.3 Comparison of methods for sensitivity integration

By inspecting Algorithm 3 it is easy to see that the total integration length is NT_s. The total integration length for Algorithm 4 is $\sum_{k=1}^{N-1} kT_s = \frac{1}{2}N(N-1)T_s$ which at first sight makes Algorithm 3 seem like a better choice. However, Algorithm 4 has the advantage that the dimension of the sensitivity ODE is only $n_x n_u$ compared to Algorithm 3 where the dimension of the sensitivity ODE is $n_x(n_x + n_u)$. In applications where the dimension of the state vector is large compared to the dimension of the control vector Algorithm 4 may outperform Algorithm 3 even though the total integration length is increased.

To demonstrate how Algorithm 3 and Algorithm 4 compare for different dimensions of the state vector, consider the one dimensional heat equation

$$\frac{\partial \Psi}{\partial t}(x,t) = \frac{\partial^2 \Psi}{\partial x^2}(x,t), \quad x \in [0,1], \quad t \geq 0. \tag{28}$$

$\Psi(x,t)$ is the temperature distribution in the medium which is assumed to have length 1. Moreover, assume that the temperature at the end of the medium is fixed to $\Psi(1,t) = 0$ while the temperature on the other end can be directly controlled, i.e. $\Psi(0,t) = u(t)$. The term $\frac{\partial^2 \Psi}{\partial x^2}$ is approximated using a second order central difference approximation. This gives a finite dimensional model where the spacial discretization length and thereby also the number of states may be increased arbitrarily.

Simulated runtime as a function of dimension of the state vector is shown in Figure 3 which demonstrates that

Algorithm 4 becomes more efficient than Algorithm 3 as the dimension of the state vector is increased. These simulations were implemented in Matlab using CVODES as solver routine.

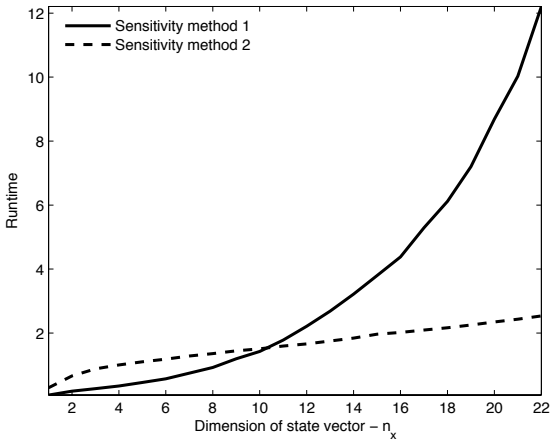

Fig. 3. Runtime - Integration of sensitivities

Note that the sensitivities calculated by Algorithm 4 needs to be combined with Algorithm 5 in order to build the impulse response matrix. Another important point to make is that Algorithm 4 incorporates most of the matrix multiplications performed by Algorithm 1 or 2 in the simulation of the ODEs. The total number of multiplications in Algorithm 5 is

$$n_z n_x n_u \sum_{k=1}^{N} k = \frac{n_z n_x n_u N(N+1)}{2}. \tag{29}$$

Comparison of (19) and (29) shows that the number of multiplications for Algorithm 5 can be small compared to Algorithm 1 and 2 when the dimension of the state vector is large, which is the second reason why sensitivity method 2 may be better suited if $n_x \gg n_u$.

Finally, we make some remarks regarding using variable-step ODE-solvers and input blocking. When using a variable step-size ODE solver, one needs to reset integration at every sample instant due to the reinitialization of S_k and W_k in Algorithm 3 and 4 (and also, ideally, due to the discontinuity introduced by the sample-and-hold implementation of the input).

This reinitialization is typically expensive in terms of computational complexity. However, when using input blocking, one can implement algorithms similar to the above such that reinitialization takes place only once per input block, giving a significant speed-up.

5. CONCLUDING REMARKS

We have investigated some issues regarding gradient computation using both the forward and adjoint techniques. To minimize the number of multiplications required for constructing Ξ the forward method should be preferred when $n_u < n_z$ and the adjoint method for the opposite case. We also demonstrated that the adjoint method can benefit further from parameterizing the output variables using $N_z < N$ coincidence points. Finally, these algorithms were discussed in combination with different approaches for integration of sensitivity equations for continuous-discrete systems.

Although the setup used might seem somewhat outdated from an academic point of view, in practice these types of algorithms have advantages that makes them amenable for industrial use (Foss and Schei, 2007; Pluymers et al., 2008). Compared to using finite differences, one can argue that using analytical methods for sensitivity integration as explored herein, has extra potential in cases where the models come from advanced modeling tools with capability of producing symbolic Jacobians (Imsland et al., 2009).

REFERENCES

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

Bock, H.G., Diehl, M., Leineweber, D.B., and Schlöder, J.P. (2000). A direct multiple shooting method for real-time optimization of nonlinear DAE processes. In F. Allgöwer and A. Zheng (eds.), *Nonlinear Predictive Control*, volume 26 of *Progress in Systems Theory*, 246–267. Birkhäuser, Basel.

Cagienard, R., Grieder, P., Kerrigan, E., and Morari, M. (2007). Move blocking strategies in receding horizon control. *Journal of Process Control*, 17, 563–570.

Cao, Y., Li, S., Petzhold, L., and Serban, R. (2003). Adjoint sensitivity analysis for differential-algebraic equations: The adjoint dae system and its numerical solution. *SIAM J. SCI. COMPUT.*, 24, 1076–1089.

De Oliveira, N.M.C. and Biegler, L.T. (1995). An extension of newton-type algorithms for nonlinear process control. *Automatica*, 31(2), 281–286.

Foss, B.A. and Schei, T.S. (2007). Putting nonlinear model predictive control into use. In *Assessment and Future Directions Nonlinear Model Predictive Control*, LNCIS 358, 407–417. Springer Verlag.

Hairer, E., Nørsett, S.P., and Wanner, G. (1993). *Solving Ordinary Differential Equations I – Nonstiff problems.* Springer-Verlag, 2nd edition.

Hindmarsh, A.C. and Serban, R. (2006). *User Documentation for CVODES v2.5.0.* Center for Applied Scientific Computing, Lawrence Livermore National Laboratory.

Imsland, L., Kittilsen, P., and Schei, T.S. (2009). Using modelica models in real time dynamic optimization – gradient computation. In *Proc. of Modelica'2009.* Como, Italy.

Jørgensen, J.B. (2007). Adjoint sensitivity results for predictive control, state- and parameter-estimation with nonlinear models. In *European Control Conference 2007.*

Li, W.C. and Biegler, L.T. (1989). Multistep, newton-type control strategies for constrained, nonlinear processes. *Chem. Eng. Res. Des.*, 67, 562–577.

Maciejowski, J.M. (2001). *Predictive Control with Constraints.* Prentice-Hall.

Pluymers, B., Ludlage, J., Ariaans, L., and Brempt, W.V. (2008). An industrial implementation of a generic nmpc controller with application to a batch process. In *Proceedings of the 17th IFAC World Congress.*

Schlegel, M., Marquardt, W., Ehrig, R., and Nowak, U. (2004). Sensitivity analysis of linearly-implicit differential-algebraic systems by one-step extrapolation. *Applied Numerical Mathematics*, 48, 83–102.