Efficient Implementation of Step Response Models for Embedded Model Predictive Control

D. K. M. Kufosalor\textsuperscript{a}, L. Imsland\textsuperscript{a}, T. A. Johansen\textsuperscript{a,b}

\textsuperscript{a}Department of Engineering Cybernetics, Norwegian University of Science and Technology (NTNU), O.S. Bragstads plass 2D N-7491 Trondheim, Norway.
\textsuperscript{b}Center for Autonomous Marine Operations and Systems, NTNU.

Abstract

This paper proposes efficient step response model implementation strategies that lead to accurate control and high computational performance in an embedded Model Predictive Control (MPC) scheme. Different implementations of the step response prediction model are examined, and inherent properties that directly affect control performance in the presence of disturbances are discussed. Model errors that are inconsistent with bias updates (i.e. the model of unknown disturbances commonly used in step response MPC) are identified, and it is shown that the bias updates may worsen the effect of the errors in some cases. Particular attention is paid to the robustness of the prediction models to small truncation errors and errors in the input or measured disturbance history. Several implementation aspects that are crucial for embedded targets with limited resources are discussed. The findings are illustrated by simple simulation examples and an industrial case-study involving hardware-in-the-loop simulation of a subsea compact separation process.

Keywords: Model Predictive Control; Step response models; Robustness to errors; Unknown disturbances; Embedded systems

1. Introduction

Model Predictive Control (MPC) is well established as the preferred method for challenging constrained multivariable control problems, especially in the process industries. Most industrial MPC solutions incorporate several features, including ad hoc modifications, that contribute to achieve satisfactory control performance for different applications (Qin and Badgwell, 2000; Maciejowski, 2002). Due to the possibility of operator intervention and the reliance on conventional low-level controllers, critical issues, such as closed-loop stability guarantee and robustness to unknown disturbances, are not the main focus in many industrial applications (Maciejowski, 2002). However, when considering safety-critical applications, Johansen (2015) points out the fact that although safe operation may be accounted for by building a resilient architecture around an MPC scheme, the effects of model uncertainty and unknown disturbances on the control performance are still fundamental limitations.

In a model-based control approach the accuracy of the model used may have a significant effect on the performance of the controller, and in the case of MPC, inaccurate predictions can lead to undesirable control performance. The type of model used also incorporates specific characteristics and capabilities to the MPC scheme. For instance, the use of step response models is limited to asymptotically stable and integrating plants, whereas a state-space or ARX model can also describe unstable systems.

According to Maciejowski (2002), Dynamic Matrix Control (DMC) (Cutler and Ramaker, 1979) and its derivative, DMC-Plus, are the most widely known and used of the commercially available MPC products, with particularly high usage in the petrochemicals sector. These MPC products use finite step response prediction models that rely on simple disturbance models and assumptions on future plant disturbance behavior (Qin and Badgwell, 2000; Maciejowski, 2002; Muske and Rawlings, 1993). Another example of industrial MPC software packages is SEPTIC (Statoil Estimation and Prediction Tool for Identification and Control) Strand and Sagli (2003), which achieves good performance for many applications by using step response models. The use of step response models is still common in industrial MPC implementations mainly because step response models are easy to build, understand, and maintain (Strand and Sagli, 2003; Lee et al., 1994). Nevertheless, the use of alternative model representations (e.g. state-space models) is possible in SEPTIC for applications where a better prediction error propagation will lead to improved control performance.

The above MPC software examples incorporate numerous features into a PC/server-based software package, in order to facilitate the achievement of high control performance for different applications. The features include strategies for correcting inaccurate predictions and, in some cases, the plant operator is able to tune parameters for compensating model errors (Strand and Sagli, 2003; Qin and Badgwell, 2000; Maciejowski, 2002). Due to the limited computational resources available on typical ultra-reliable industrial hardware (e.g. PLCs), and limited means and opportunity for operator intervention, all the features that are found in industrial MPC software packages cannot always be expected to fit into an embedded MPC framework.
Johansen (2015). For this reason, the enhancement of MPC performance in a relatively simple framework is examined in this paper.

A known issue of step response models is the large amount of data usually required in order to produce accurate enough predictions in MPC. The amount of data tends to be very large for applications where fast sampling rates are necessary for desired control performance targets (Lundström et al., 1995; Hovd et al., 1993; Maciejowski, 2002). Due to practical limits on the amount of step response data that can be used in an application, truncated models may be considered. Nevertheless, the extent of truncation is limited. In fact, large truncation errors may not only lead to poor performance, but also instability (Lundström et al., 1995). Different techniques have been introduced to capture the residual neglected when the step response sequence is truncated (see for example Hovd et al. (1993); Lee et al. (1994)). However, such techniques tend to incorporate other types of model representations, leading to ‘hybrid’ formulations, and may introduce complications that reduce the main attractiveness of step response models. This paper differs from the work described above in that the main goal is to enhance the efficiency of prediction models that rely only on step response data. Different approaches are examined, providing straightforward comparisons and results that will benefit an industrial MPC design and implementation process. Moreover, the model errors examined in this paper are those incurred due to common practice (e.g. small truncation errors), and the effect of such errors may be more severe in a relatively simple MPC framework for embedded applications.

This paper extends the material in Kufoalor et al. (2015c,d), where different step response model implementations found in the existing MPC literature are categorized into two main approaches. The approaches are referred to as the standard and recursive implementations. Some properties that distinguish the strategies used in both implementations were analyzed in Kufoalor et al. (2015d), and different prediction quality enhancement strategies were presented in both papers. However, the implementation aspects treated partly in Kufoalor et al. (2015c,d) are related, and a more comprehensive discussion is achieved when the different aspects are treated together. The outcome is presented in this paper. Moreover, this paper presents further details and new findings. In particular, the model errors discussed may lead to effects that are inconsistent with the use of bias updates (i.e. the model of unknown disturbances commonly used in step response MPC, described in Section 2.1). This paper therefore highlights the combined effect of model errors and bias updates.

2. Step response prediction models

The step response model:

\[ y(k) = \sum_{i=1}^{N-1} s(i) \Delta u(k - i) + s(N) u(k - N) \]  

(1)

can be used to describe the dynamic and static interactions between the input \( u \) (also known as manipulated variable or MV) and the output \( y \) (known as the controlled variable or CV) of an asymptotically stable single-input-single-output (SISO) system, where \( \Delta u(k) = u(k) - u(k-1) \). The step response coefficients \( s(i) \) reach constant values after \( N \) sampling periods, such that for large enough \( N, s(N+1) \approx s(N) \). For simplicity, implementations for SISO systems are discussed first.

Based on (1), a prediction of the future output trajectory can be computed using

\[ \hat{y}(k+j|k) = \sum_{i=1}^{j} s(i) \Delta u(k+j-i) + \sum_{i=j+1}^{N-1} s(i) \Delta \hat{u}(k+j-i) + s(N) \hat{u}(k+j-N) + v(k|j) \]  

(2)

(see e.g. Garcia et al. (1989)). The notation \( \hat{y}(k+j|k) \) represents the prediction of \( \hat{y}(k+j) \) using available information at time \( k \). The first line on the right-hand side of (2) consists of known terms, including the present and future input moves \( \Delta \hat{u}(\cdot) \), while the second line contains known terms computed using the past input \( \hat{u}(\cdot) \), past input moves \( \Delta \hat{u}(\cdot) \), and a disturbance model \( v(k|j) \).

2.1. Disturbances

Without output feedback, the cumulative effects of unmeasured disturbances and model errors will lead to inaccurate predictions. A disturbance model \( v(k+j|k) \) is therefore used:

\[ \hat{y}(k+j|k) = \bar{y}(k+j|k) + v(k+j|k), \]  

(3)

\[ v(k+j|k) = v(k|k) = y_m(k) - \bar{y}(k|k-1), \]  

(4)

where \( y_m \) is the measured plant output at time \( k \). The model (4) is a usual choice known as a bias term used in correcting \( \bar{y}(k+j|k) \), and it provides integral action in MPC (see e.g. Muske and Rawlings (1993); Maciejowski (2002)). Throughout this paper, the notation \( \bar{y} \) is used on output vectors to indicate that the predictions are not corrected, and the use of the notation \( \bar{y} \) implies that a bias correction is applied. The simple bias term (4) assumes that an additive (step) disturbance acts on the plant output, and the disturbance remains constant for \( j = 1, \ldots, H_p \).

If this assumption does not hold in a given situation, the output prediction will be incorrect, and it may lead to poor control performance.

If a disturbance variable \( d \) can be measured, a disturbance term that contains the step response model relating the measured disturbance to each controlled variable can be added to the prediction model. Since the future changes in disturbance are typically not known at the current time \( k \), a usual assumption is that \( \Delta d(k+j) = 0, \) for \( j \geq 1 \) (see e.g. Lee et al. (1994)).

2.2. Free response

In (2), the known terms of \( \bar{y}(k+j|k) \) define the predicted unforced response of the plant \( \bar{y}_f(k+j|k) \), also known as the free response. Therefore, (2) with (3) and (4) can be written as

\[ \hat{y}(k+j|k) = \sum_{i=1}^{j} s(i) \Delta u(k+j-i) + \bar{y}_f(k+j|k), \]  

(5)
where
\[
\hat{y}_f(k + j|k) = \sum_{i=j+1}^{N-1} s(i)\Delta\tilde{u}(k + j - i) + s(N)\tilde{u}(k + j - N) + \\
y_m(k) - \sum_{i=1}^{N-1} s(i)\Delta\tilde{u}(k - i) - s(N)\tilde{u}(k - N),
\]
(6)

In other words, \(\hat{y}_f(k + j|k)\) is the response at each point along the prediction horizon, if the future inputs remains the same as \(\tilde{u}(k-1)\). Note that the remaining part of \(\hat{y}(k + j|k)\), which contains the unknown inputs, defines the forced response of the plant.

### 2.3. Matrix-vector formulation

The step response prediction model (5) can be written in a matrix-vector form by considering the following definitions:

\[
\begin{align*}
\hat{Y}(k + 1) &= [\hat{y}(k + 1|k) \ldots \hat{y}(k + H_p|k)]^T \\
\hat{Y}_f(k + 1) &= [\hat{y}_f(k + 1|k) \ldots \hat{y}_f(k + H_p|k)]^T \\
\Delta U(k) &= [\Delta u(k) \ldots \Delta u(k + H_u - 1)]^T
\end{align*}
\]

(7a)

(7b)

(7c)

The prediction models will then take the (explicit) form:

\[
\hat{Y}(k + 1) = \hat{\Theta} \Delta U(k) + \hat{Y}_f(k + 1),
\]

(8)

where

\[
\hat{\Theta} = \begin{bmatrix}
s(1) & 0 & \ldots \\
s(2) & s(1) & \ldots \\
\vdots & \vdots & \ddots \\
s(H_p) & \ldots & s(H_p - H_u + 1)
\end{bmatrix}.
\]

(9)

A matrix-vector free response \(\hat{Y}_f(k + 1)\) can be derived based on (6). However, although the currently existing step response model implementations compute the forced response in the same way (i.e. output predictions are computed explicitly), the implementations differ in the way \(\hat{Y}_f(k + 1)\) is computed. Therefore, the following sections focus on different ways of computing the free response.

### 2.4. Standard implementations

In this paper, step response prediction models that are implemented using either (6) or a similar computational approach are categorized as standard implementations. Three variants are presented in this section.

#### 2.4.1. Model A

A step response prediction model that directly implements the free response (6), i.e.

\[
\hat{y}_f^A(k + j|k) = \sum_{i=j+1}^{N-1} s(i)\Delta\tilde{u}(k + j - i) + s(N)\tilde{u}(k + j - N) + v(k|k),
\]

(10)

where \(v(k|k) = y_m(k) - \sum_{i=1}^{N-1} s(i)\Delta\tilde{u}(k - i) - s(N)\tilde{u}(k - N)\), is referred to as model A.

#### 2.4.2. Model B

An alternative formulation to (10) is derived in Maciejewski (2002):

\[
\hat{y}_f^B(k + j|k) = \sum_{i=j+1}^{N} s(i)\Delta\tilde{u}(k + j - i) + s(j)\tilde{u}(k - 1) + v(k|k),
\]

(11)

where only the past control input \(\tilde{u}(k - 1)\) is used, instead of \(\{\tilde{u}(k + j - N), j = 1, \ldots, H_p\}\), as stated in (10). Therefore, (11) avoids saving or recovering all past inputs. The standard implementation approach (11) is referred to as model B.

#### 2.4.3. Model C

Using (6), another approach is derived in Camacho and Bordons (2007): considering \(N > H_p\), the free response can be written in a more compact form,

\[
\hat{y}_f^C(k + j|k) = y_m(k) + \sum_{i=1}^{N} (s(i + j) - s(i))\Delta\tilde{u}(k - i)
\]

(12)

where \(s(i + j) - s(i) \approx 0\), for \(i > N\), has been used. This implementation approach is termed model C. Note that the constant disturbance model \(v(k|k)\) is implicitly embedded in (12), and not an explicit term as in models A and B. Similar to model B, model C uses only past input moves and avoids saving or recovering all past inputs.

Due to the compactness of (12), a simple matrix-vector model of the form

\[
\hat{Y}_f^C(k + 1) = \omega_f \hat{X}(k),
\]

(13)

can be derived, where

\[
\hat{X}(k) = [y_m(k) \Delta\tilde{u}(k - 1) \ldots \Delta\tilde{u}(k - N + 1)]^T.
\]

(14)

and \(\omega_f\) can be extracted from (12) by direct inspection:

\[
\omega_f =
\begin{bmatrix}
1 & s(2) - s(1) & \ldots & s(N) - s(N - 1) \\
1 & s(3) - s(1) & \ldots & s(N) - s(N - 1) \\
\vdots & \vdots & \ddots & \vdots \\
1 & s(1 + H_p) - s(1) & s(2 + H_p) - s(2) & \ldots & s(N) - s(N - 1)
\end{bmatrix}.
\]

(15)

Note that since \(\hat{Y}_f(k + 1)\) is the response when no future control moves are applied, it depends on the state of the plant, and \(\hat{X}\) provides a convenient way of defining the state in the standard implementation (Camacho and Bordons, 2007).

#### 2.5. Recursive implementation (Model D)

Another convenient choice of state variables that leads to a more efficient computation of MPC predictions is the current and future outputs \(\{\hat{y}(k + i), i = 0, \ldots, N - 1\}\). This choice follows the idea of Li et al. (1989) and the general extensions and variants formulated by Lee et al. (1994); Lundström et al. (1995). Given the current output trajectory prediction

\[
\hat{Y}(k) = [\hat{y}(k|k) \hat{y}(k + 1|k) \ldots \hat{y}(k + N - 1|k)]^T
\]
the goal of the model will be to compute the new output trajectory
\[
\hat{Y}(k+1) = [\hat{y}(k+1|k+1) \ldots \hat{y}(k+N|k+1)]^T
\]
when a change in input \( \Delta u(k) \) is applied. It can be shown that the entire trajectory can be computed recursively using
\[
\hat{Y}(k+1) = \mathcal{A}\hat{Y}(k) + \mathcal{B}\Delta u(k), \quad (16a)
\]
\[
\hat{y}(k) = \mathcal{C}\hat{Y}(k), \quad (16b)
\]
where \( \hat{Y}(k) \) is the state, \( \mathcal{C} = [1 \ 0 \ \ldots \ 0]_{1 \times N} \), \( \mathcal{A} \), \( \mathcal{B} \), and \( \mathcal{C} \) facilitates a very efficient implementation. Using (18), the prediction model becomes
\[
\hat{Y}(k+1) = \mathcal{A}\hat{Y}(k) + \mathcal{B}\Delta \hat{u}(k-1), \quad (17a)
\]
\[
\hat{y}(k) = \mathcal{C}\hat{Y}(k), \quad (17b)
\]
and the free response (including bias correction) becomes
\[
\hat{Y}_f(k+1) = \mathcal{A}\hat{Y}_f(k) + \mathcal{B}[1_{N}(k) - \hat{y}(k)], \quad (18)
\]
where \( \mathcal{A}\hat{Y}_f(k) \) contains the first \( H_p \) rows of \( \mathcal{A} \), and \( 1 \) is an \( H_p \)-dimensional vector which has all its elements equal to one. The special sparse structure of \( \mathcal{A}, \mathcal{A}\hat{Y}_f \), and \( \mathcal{C} \) facilitates a very efficient implementation. Using (18), the prediction model becomes
\[
\hat{Y}(k+1) = \hat{\Theta}\Delta u(k) + \mathcal{A}\hat{Y}_f(k) + 1_{N}[y_m(k) - \hat{y}(k)], \quad (19)
\]
This implementation approach is referred to as model D.

### 2.6. MIMO models

A straightforward generalization of the models to multiple-input multiple-output (MIMO) systems can be derived for all the implementation approaches. The details for each approach, including the implementation of measured disturbances (described in Section 2.1), are provided in Appendix A.

### 3. MPC problem formulation

The MPC problem is formulated as:
\[
\begin{align*}
\min_{\mathcal{U}_k} & \sum_{j=0}^{H_o-1} \|y(k+j|k) - r_s(k+j)|_2^2 + \sum_{j=0}^{H_e-1} \|\Delta u(k+j)\|_2^2 \\
+ & \rho_h\bar{e}_h + \rho_l\bar{e}_l 
\end{align*}
\]
subject to
\[
\begin{align*}
\Delta u & \leq \Delta u(k+j) \leq \bar{\Delta} u, \quad u \leq u(k+j) \leq \bar{u}, \quad (20b) \\
y - \bar{e}_l & \leq y(k+j|k) \leq \bar{y} + \bar{e}_s, \quad \bar{e}_s \geq 0, \quad \bar{e}_l \geq 0, \quad (20c) \\
u(k+j) & = u(k+j-1) + \Delta u(k+j), \quad (20d) \\
y(k+j|k) & = \sum_{i=1}^j s(i)\Delta u(k+j-i) + \hat{y}_f(k+j|k), \quad (20e)
\end{align*}
\]
where \( j \in \{H_o, \ldots, H_p\}, \quad H_o > 1, \) for the output constraints, and \( j \in \{0, \ldots, H_e-1\} \) for the input constraints. The reference is denoted by \( r_s(k+j) \). The slack variables \( \bar{e}_h, \bar{e}_l \), weighted by \( \rho_h, \rho_l > 0 \), relax the constraints of the CV to avoid infeasibility in case of large disturbances or prediction model errors. Nominal closed-loop stability can be achieved by an adequate choice of the weights \( \bar{Q}_s, \bar{P} \), and the horizon lengths \( H_o, H_e \) (see e.g. Grüne and Pannek (2011)). Note that the MPC problem is formulated using a SISO system description. An extension to MIMO systems is made in the following description.

Problem (20) can be converted into a sparse quadratic programming (QP) problem (see e.g. Maciejowski (2002); Kufaolor et al. (2014)), where structure exploiting solvers can be used to compute the solution. An alternative is to use the equality constraints for elimination to produce a more dense QP as presented in Kufaolor et al. (2015b). Since the choice of QP solver is not the main focus of this paper, only the sparse formulation is presented:
\[
\begin{align*}
\min_{Y} & \quad Y^TQ_sY + \Delta U^TP\Delta U - 2\mathcal{T}^TQ_sY + \rho_h^T\bar{e}_h + \rho_l^T\bar{e}_l \\
\text{subject to} & \quad E\Delta U \leq e, \quad FU \leq f, \quad (21a) \\
& \quad GY - M_s\bar{e}_h - M_l\bar{e}_l \leq g, \quad \bar{e}_h \geq 0, \quad \bar{e}_l \geq 0, \quad (21c) \\
& \quad KU - \Delta U = \hat{\Gamma}\hat{u}(k-1), \quad (21d) \\
& \quad Y - \Theta\Delta U = \bar{Y}_f, \quad (21e)
\end{align*}
\]
where the vectors and matrices involved apply to MIMO systems according to the definitions in Kufaolor et al. (2015a), which are based on the notations of Maciejowski (2002). \( \mathcal{T} \) represents the reference vector, and it is straightforward to deduce the constraint matrices.

The paper focuses on the computation of the term \( \bar{Y}_f \) in (21e), which represents the free response prediction model. It is the only variable/vector that is different for a given prediction model implementation in the MPC problem. All the remaining matrices and vectors in (21) are the same for the different prediction model implementations. The free response \( \bar{Y}_f \) is computed only once at each sampling time \( k \), and the predictions enter (21e) as parameters. In order to reduce the number of decision variables, move blocking is implemented and CVs are evaluated on only specified evaluation points on the prediction horizon (see Kufaolor et al. (2015a) for further details).

### 4. Efficient implementation strategies

#### 4.1. Computational efficiency

The differences in computational efficiencies were investigated by Li et al. (1989), where the main focus was the diff-
ference in computational speed. Based on the comparison of the number of floating point operations involved, the recursive implementation turns out as the most computationally efficient approach. In this work, the difference in memory usage is also emphasized as a key factor to consider for embedded hardware with limited memory/storage resources.

The free response of model D has a special sparse structure that can be exploited to enhance its computational efficiency. Note that the $A_p$ matrix in (A.6) has only $nCV \cdot N$ nonzero elements out of $(nCV \cdot N) \times (nCV \cdot N)$ elements, and $A_p$ has only $nCV \cdot H_p$ nonzero elements out of $(nCV \cdot H_p) \times (nCV \cdot N)$ elements. Therefore, an implementation that does not exploit the sparse structure will be inefficient for an embedded system with limited resources. In this work, code generation is used where sparse matrix storage formats and sparse matrix-vector operations are exploited to keep memory usage as low as possible. The terms containing the sparse matrices can be prepared offline for fast online evaluation, involving only the nonzero terms of the matrices. Using the above strategies lead to code sizes for model D that are usually small compared to models A, B, and C.

4.2. Computational enhancement strategies

A particular model (A, B, C, or D) may be preferred based on the convenience or consequences associated with the implementation aspects. Therefore, specific computational properties and implementation strategies are discussed in this section. The MIMO models in Appendix A are used in the discussions in order to fully capture the implementation aspects in a realistic way.

Model B (A.4) is simpler to implement, and is computationally more efficient compared to variant A (A.3), since it avoids either saving $(n_{MV} + n_{PB})H_p$ past inputs and disturbances, or computing these values at each sampling time $k$ from $\tilde{u}(k-1)$ and $\Delta \tilde{u}(k)$. However, it is shown later (in Section 6.2) that model B requires further enhancements that lead to computations and memory usage similar to that of model A. Model C (A.5) also avoids saving or recovering all past inputs and disturbances, but at the expense of extra computations that are introduced through the extension made using the assumption that $s(N+i) - s(N) \approx 0$, for $i \geq 1$. Therefore, in the case where memory is the main limiting factor on a target embedded hardware, model C will be an appropriate choice compared to model A. Otherwise, model A is a better option (compared to model C) when speed is the limiting factor. In the case where both memory and computational speed are crucial, model D will be the most appropriate option.

A common disadvantage of models A and B is that they depend explicitly on the extra bias term $V(k)$, which involves the computation of the predicted outputs $\tilde{y}(k|k-1)$. In general, this term is not directly accessible from any of the remaining terms of the free response computations. Note that the computation of $\tilde{y}(k|k-1)$ involves the full step response model data and all the available past input moves and past changes in measured disturbances (see (6)). However, the extra computation can be avoided in some cases. If $H_w = 1$, then $\tilde{y}(k|k-1)$ can be calculated by

$$\tilde{y}(k|k-1) = \tilde{y}(k|k-1) - v(k|k-1).$$

This is not possible if $H_w > 1$. The models C and D, on the other hand, avoid the extra calculations associated with the bias term $V(k)$, and therefore have a clear computational advantage over variants A and B, when (22) cannot be used.

Despite the convenience of the matrix-vector formulations for MIMO systems, it may be necessary to extend some of the step response data when implementing the MIMO models. For models A, B, and C, the number of samples $N_j$ defined as the largest $N$ of the SISO step response models corresponding to MV number $j$, can be used. On the other hand, a corresponding definition $N_i$ can be used for variant D, where $N_i$ is the largest $N$ of the SISO step response models corresponding to CV number $i$. However, the use of offline pre-computations and code generation facilitate the use of distinct $N_j$ values for the models of each MV $- CV_i$ pair (and similar for DV – CV models).

The use of MV blocking, which introduces non-uniform intervals between control decisions, and a strategy to reduce the number of CV evaluation points in the MPC problem can also be considered to speed up the computations and further reduce memory usage of step response models.

5. Inherent model properties and error effects

The simple bias compensation used in the prediction models assumes that the unknown disturbances are constant. It is known that model errors such as errors in the steady-state gain fall within this category of errors (Maciejowski, 2002). The effects of large truncation errors in the step response data are also well studied (see e.g. Lundström et al. (1995)). However, small truncation errors, for which $s(N+i) \approx s(N)$, $i \geq 1$, is valid, are usually not considered as a significant source of (unknown) disturbances in step response prediction models. In fact, due to measurement noise and other practical factors, obtaining a step-response data set with $s(N+i) = s(N)$ is not a usual goal in practice.

It is common practice to extend the step response data in order to either achieve appropriate dimensions in a matrix-vector formulation for MIMO systems (see e.g. Maciejowski (2002)) or to obtain compact variants of model A, as in model C (Camacho and Bordons, 2007). The way the truncated (and extended) step response data is used in the model may introduce different error effects, which must be compensated for. Therefore, this section examines inherent properties of the prediction models, considering different error effects. The discussions are extended to the effect of unknown errors in the state vector of step response models (i.e. a suitable vector of past inputs and outputs), and an example that illustrates the combined effect of unknown disturbances and bias updates is presented.

For simplicity, general properties are discussed by referring to the models as either standard or recursive implementations, and comments specific to a particular model (i.e. A, B, C, D) are provided where necessary.
5.1. Effect of small truncation errors in the recursive implementation

The propagation of truncation errors through the recursive implementation is discussed in this section. It is shown that the bias estimate is unable to accurately capture unknown disturbances unless the effect appears in the output prediction. That is, model errors that affect future predictions before affecting the computation of the current output, lead to faulty input moves being calculated based on the erroneous predictions.

Consider a single-step modification of the step response data of the recursive implementation, using \( s(N - 1) \) instead of \( s(N) \), i.e.

\[
\{s(1), s(2), \ldots, s(N - 1), s(N - 1)\}. \tag{23}
\]

This simple modification introduces a ‘small’ truncation error \( e_N = s(N) - s(N - 1) \) into the step response data. Using (23) in the recursive implementation implies that the last element of \( \tilde{\mathbf{B}} \) is replaced by \( s(N - 1) = s(N) - e_N \) (cf. (17)). Due to the structure of \( \tilde{\mathbf{B}} \), it is straightforward to verify that, whenever \( \Delta \tilde{u}(k - 1) \neq 0 \), the last element \( \tilde{y}(k + N - 1|k) \) of the computed output prediction \( \tilde{Y}(k) \) will experience a step error of magnitude \( -e_N \Delta \tilde{u}(k - 1) \).

In order to trace the propagation of a single truncation error through the free response model, assume that the error is activated at time \( k = 1 \) by \( \Delta u(0) \), and no further input moves are made until \( k = N - 1 \). The propagation of the error through the state vector \( \tilde{Y}(k) \) from \( k = 1 \) to \( k = N \), using (17), can be written out as follows:

\[
\begin{align*}
\tilde{y}(1|1) & = 0 \\
\tilde{y}(2|1) & = 0 \\
\vdots & \\
\tilde{y}(N-1|1) & \\
\tilde{y}(N|1) - e_N \Delta \tilde{u}(0) & \\
\end{align*}
\]

\[
\begin{align*}
\tilde{y}(1|1) & = 0 \\
\tilde{y}(2|1) & = 0 \\
\vdots & \\
\tilde{y}(N|1) - e_N \Delta \tilde{u}(0) & \\
\tilde{y}(N|1) - e_N \Delta \tilde{u}(0) & \\
\end{align*}
\]

\[
\begin{align*}
\tilde{y}(1|1) & = 0 \\
\tilde{y}(2|1) & = 0 \\
\vdots & \\
\tilde{y}(N|1) - e_N \Delta \tilde{u}(0) & \\
\tilde{y}(N|1) - e_N \Delta \tilde{u}(0) & \\
\end{align*}
\]

\[
\begin{align*}
\tilde{y}(1|1) & = 0 \\
\tilde{y}(2|1) & = 0 \\
\vdots & \\
\tilde{y}(N|1) - e_N \Delta \tilde{u}(0) & \\
\tilde{y}(N|1) - e_N \Delta \tilde{u}(0) & \\
\end{align*}
\]

Note that at time \( k = N - 1 \), the error has not yet affected the current output prediction, which in this case is equal to \( \tilde{y}(N - 1|1) \). The bias will be \( y_m(N - 1) - \tilde{y}(N - 1|1) = 0 \), assuming noise-free measurements. However, the prediction of the future output (e.g. one-step ahead) will be wrong, and an erroneous input move will be computed at \( k = N - 1 \) (cf. (19)):

\[
\tilde{y}(N|N - 1) = s(1) \Delta u(N - 1) + \tilde{y}(N|1) - e_N \Delta u(0).
\]

Obviously, the prediction error magnitude \( -e_N \Delta u(0) \) determines the amount of erroneous control action \( \Delta u(N - 1) \) computed. Therefore, the truncation error should be ‘sufficiently small’, such that the false control moves that occur due to \( -e_N \Delta u_{\text{max}} \) are insignificant considering the accuracy requirements of the closed-loop control.

At time \( k = N \), the truncation error finally appears in the current output prediction, and therefore captured in the bias as

\[
y_m(N) - \tilde{y}(N|1) + e_N \Delta \tilde{u}(0) - s(1) \Delta \tilde{u}(N - 1) = e_N \Delta \tilde{u}(0).
\]

Since the same bias is applied to all future output predictions (see (19)), the predictions will be free from the truncation error effect activated at time \( k = 1 \). However, each time a new input move is made (i.e. \( \Delta u(k) \neq 0 \)), a corresponding step error (of size \( -e_N \Delta u(k) \)) will be made in the output prediction \( N - 1 \) time steps in the future.

In short, the state vectors (for \( k = 1, \ldots, N \)) show that a step (or an additive) error that enters the state vector \( \tilde{Y}(k) \) will be propagated forward in \( \tilde{Y}(k) \) from one sampling time to the next. If the error affects the last element of \( \tilde{Y}(k) \), it will remain part of \( \tilde{Y}(k) \) for all future time (as in the case of the truncation error effect). This is because the last element in \( \tilde{Y}(k) \) is repeated after each time shift. Once the error appears in the current output prediction, the bias update will compensate for that particular error correctly. However, faulty input moves will be computed before the error eventually affects the current output prediction.

5.2. Effect of small truncation errors in the standard implementation

The propagation of truncation errors through the standard implementation is fundamentally the same as the recursive implementation. This can be easily verified by examining the state vector \( \tilde{X} \) and matrix \( \tilde{A}_r \) (see (14) and (15)). It can be seen from \( \tilde{A}_r \) that a change in input \( \Delta \tilde{u}(0) \) will affect the last column of \( \tilde{A}_r \) after \( N - 1 \) sampling intervals. If the truncated data (23) is used, the last column elements will become \( s(N) - e_N - s(N - 1) \). This means that an erroneous prediction step of \(-e_N \Delta \tilde{u}(0)\) occurs exactly as in the recursive implementation. The error enters the current output computation \( \tilde{y}(k|k) \) at the next sampling interval, and therefore a correct bias update for \( \tilde{y}(k + 1|k) \) will be computed.

Nevertheless, the way the truncated step response data is used in \( \tilde{A}_r \) indicates that, if the input keeps changing, the magnitude of the bias will not be adequate for correcting the error incurred at all points in the prediction horizon. The structure of the error introduced into \( \tilde{A}_r \) can be derived by using \( s(N - 1) = s(N) - e_N \) to replace \( s(N) \) in (15):

\[
\begin{bmatrix}
    -e_N & 0 & \cdots & \cdots & \cdots \\
    \vdots & \ddots & \ddots & \ddots & \cdots \\
    \vdots & \cdots & \ddots & \cdots & \cdots \\
    \vdots & \cdots & \cdots & -e_N & \cdots \\
    \vdots & \cdots & \cdots & \cdots & -e_N
\end{bmatrix}
\Delta \tilde{u}(k + H_p - N) = 0
\]
5.3. Effect of errors in state vectors and bias updates

Different definitions of the state in step response models are presented in sections 2.3 and 2.5. Errors in the state vectors exhibit a similar effect as that discussed in Section 5.1. That is, a combined effect of model errors and wrong bias updates. However, the bias updates may worsen the effect of the model errors in some cases.

Since the state of the recursive implementation consists of the predicted current and future outputs, the propagation of unknown errors through the state vector is the same as discussed in Section 5.1. On the other hand, the standard implementation relies on historic data to compute the state. Therefore, a different error effect may occur in the standard implementation. Examples include initialization errors and unknown disturbances that affect the input or measured disturbance history in such a way that the plant state becomes inconsistent from one sampling instant to the next. For such errors, it may take several time steps before the effect appears in the output prediction used to compute the bias estimate. Moreover, since the affected past data will be shifted through the state vector, the error may lead to adverse effects at different stages into the future.

A simple example (presented in the next section) is used to show that when the state in the standard implementation is affected by an unknown error, the bias updates may contribute significantly to the deterioration of the model’s prediction capabilities. Since a different state vector is used in the recursive implementation, the outcome is not the same as that of the standard implementation. Nevertheless, the discussions in Section 5.1 show that the adverse effect of bias updates due to unknown errors in the state vector may degrade the performance of the recursive implementation as well.

5.4. Example 1: effect of errors in data history

Consider the simple mass-damper-spring system:

\[
\begin{bmatrix}
    x_1(k+1) \\
    x_2(k+1)
\end{bmatrix} = \frac{1}{10} \begin{bmatrix}
    8 & 6 \\
    -3 & 2
\end{bmatrix} \begin{bmatrix}
    x_1(k) \\
    x_2(k)
\end{bmatrix} + \begin{bmatrix}
    4 \\
    6
\end{bmatrix} u_p(k) + \begin{bmatrix}
    0.4 \\
    0.6
\end{bmatrix} d(k),
\]

\[
y_1(k) = x_1(k) + w_1, \quad y_2(k) = x_2(k) + w_2, \quad u_p(k) = u(k) + \omega.
\]

The system (25) is asymptotically stable, controllable and observable. There is a known disturbance variable \(d\), which represents a persistent force, and unknown disturbances, affecting the plant input \(u_p\) (i.e. \(\omega\)) and outputs (i.e. \(w_1\) and \(w_2\)). The sampling time of the system is 1s, and the step responses are shown in Fig. 1. The state-space model (25) is used to simulate the plant, and the step response data is used in the prediction models. For closed-loop performance analyses, the MPC scheme (20) is used, where \(H_u = H_y = 12\), \(u = -2.5\), \(\bar{u} = 10\), \(y = [0, -10]^T\), \(\bar{y} = [20, 10]^T\), \(\bar{Q}_y = \text{diag}(1, 0)\), and \(\bar{P} = 1\). The penalty matrix \(\bar{Q}_y\) indicates that changes in reference \(r_y\) are made for only \(y_1\).

The effect of errors in the state vectors and the adverse effects of bias updates are illustrated in the simulation example shown in Fig. 2. The MPC setup described above with \(N = 14\) was used, and a unit error was simulated in the measured disturbance history. The simulation is initialized with the plant at rest, and the past disturbance data \(\bar{d}(k-1)\) is affected by the unit error at time \(k = 2\). The output reference was set to \(r_{y_1} = 0\), and no other errors were simulated in the entire duration of the simulation. The test setup makes it easy to trace the movement of a single past data point. Note that even though the error affects both the standard and recursive implementations through the use of \(\Delta d(2) = -1\), the recursive implementation does not rely on historic data from several time intervals in the past.

As discussed in sections 5.1–5.3, it is clear from Fig. 2 that an erroneous input move drives the plant output away from zero (one step) before the effect of the error starts showing up in the bias estimate (at time \(k = 3\)). A wrong prediction was made due to the bias update, which led to the erroneous input move. Note that the full effect of the error does not appear on the output after only one step. This is because the model of unmeasured disturbances describes a disturbance at the output, while this...
error enters as a disturbance at the input. Consequently, it may take several sampling intervals before the full effect is observed on the output. A step error in an input may therefore lead to a ‘ramp-like’ behavior on the output, as seen in Fig. 2.

Observe also that when suitable bias updates are made, adequate input moves are computed to drive the plant output back to the setpoint. However, in the case of the standard implementation, the error is propagated through vector \( \Delta D(k) \) as \( -1 \) and vector \( D(k - N) \) as \( 1 \). Specifically, the error enters the standard implementation’s disturbance history at time \( k = 3 \) as (cf. Appendix A.1)

\[
\hat{D}(k - N) = [0, \ldots, 0, 1]^T_{1 \times 12}, \quad \Delta D(k) = [−1, 0, \ldots, 0]^T_{1 \times 12}.
\]

This introduces inconsistencies in the disturbance history that are not handled well by bias updates. The error effect does not reflect in the bias estimates since the error in \( \hat{D}(k - N) \) does not affect the computation of the current output \( \hat{y}(k|k - 1) \). The adverse effect of the error becomes increasingly significant as the error is shifted through the vectors, leading to the erroneous input moves observed after time \( k = 9 \). When the error reaches the beginning of vector \( \hat{D}(k - N) \) at time \( k = 14 \), its effect cancels out the effect of the error in vector \( \Delta D(k) \). At this stage, the prediction \( \hat{y}(15|14) \) is not affected by the error, meaning that the bias estimate will be zero at the next time step, \( k = 15 \) (see Fig. 2). Consequently, a large input move is computed at \( k = 15 \). At the same time, the error is shifted out of \( \hat{D}(k - N) \) and \( \Delta D(k) \), and it becomes possible to compute adequate input moves to drive the plant back to the setpoint.

Recall that the disturbance history is part of the past data used to compute the plant state in the standard implementation. Therefore, this example shows that the bias updates are unable to properly compensate for errors that lead to inconsistent past data. The predictions become worse due to the resultant effect of the error and the bias updates. In this case, a ‘state disturbance model’ will be more appropriate for the standard implementation.

6. Prediction quality enhancement strategies for the standard implementations

In this section, the discussions focus on strategies that aim at enhancing the standard implementations in order to avoid cumulative error effects. Cumulative error effects occur when model errors are inconsistent with the bias updates and the resulting erroneous control moves increase in magnitude over time. This effect deteriorates the prediction capability of the model. An example that discusses the consequence of cumulative error effects is presented in the next section, followed by specific proposals that can be implemented in order to avoid such error effects. A compact implementation based on model A is also derived, where the strategy used ensures that cumulative error effects are not introduced into the model.

6.1. Example 2: destabilizing effect of errors in the standard implementations

The simple plant (25) and the MPC parameters in Example 1 are used in this example. Inspection of the step response coefficients (plotted in Fig. 1) reveals that the step response sequence from \( u \) to \( y_1 \) reaches a steady value at 19 samples, whereas the response from \( u \) to \( y_2 \) settles to a constant value from sample 22. If the exact steady values are used, \( s(N + i) \) will be equal to \( s(N) \), for \( i \geq 1 \). However, by inspection (see Fig. 1), the sample number 12 provides steady-state values close to the true steady-state. Therefore, sample 12 is an acceptable point to truncate all the step response sequences. Fig. 3 shows the consequence of the cumulative effect of small truncation errors.

The plots of Fig. 3 are obtained when \( s(12) \) is used to extend the step response data to \( N = 13 \), and a modified version of model A is implemented. That is, the case where the zeros in \( \Psi_{1, j}^A \) (cf. Appendix A.1) are replaced by \( s(N_j) \) based on the assumption \( s(N + 1) \approx s(N) \). The errors introduced into the model result in a cumulative effect that becomes significant over time. Although the system is initialized at the setpoint (where the system is at rest), erroneous input moves are computed, and the effect is gradually amplified. The results in Fig. 3 represent the case where no bias correction is applied. However, a gradual buildup can be seen in the bias estimates as well. In the presence of disturbances or a setpoint change, the bias builds up much faster, and applying feedback through the use of bias updates makes the situation worse. Similar destabilizing effects are obtained when model B is used in the simulation, either without or with a modified \( \Psi_{1, j}^B \) (as in Maciejowski (2002)). This is because model errors that result in a cumulative effect are present in model B. Therefore, an enhancement is proposed for model B in the next section.

Note that the cumulative error effects, due to truncation errors, depend on the prediction horizon \( H_p \). Therefore, the error effects become more significant for large \( H_p \), especially in the case where \( N - H_p \) is small. This may lead to a challenging MPC design and tuning process for some applications. Recall that industrial MPC solutions are typically implemented at relatively slow sampling rates, and at a high control level (above basic dynamic controllers) (see e.g. Qin and Badgwell (2000)).
Therefore, there is usually ample time to apply some ad hoc measures for verifying and handling control effects that become worse over time. However, this may not be the case in embedded applications where cumulative error effects are not introduced into the implementation that has similar computational advantages compared to model B if the enhancement in Section 6.2 is implemented.

6.2. Enhancing model B

Model errors, inconsistent with the bias updates are identified when the derivation of model B is inspected. Compared to model A, the residual ignored in model B is

\[ y_{res}(k + f) = s(N)\tilde{u}(k + j - N) - s(N)\Delta\tilde{u}(k + j - N) - s(j)\tilde{u}(k - 1). \]  
(26)

The derivation of (26) is given in Appendix B. This can be easily verified by adding (26) to (11), which becomes equal to (10). The residual ignored in model B is not constant over the prediction horizon, and cannot be properly handled by a simple bias correction. Therefore, (26) should be included in the implementation of model B in order to avoid a cumulative effect that leads to significant errors in the predictions.

6.3. Avoiding cumulative effects of small truncation errors

Apart from ensuring that the truncation error is sufficiently small, specific strategies can be implemented to ensure that the error effect (24) becomes insignificant (or not amplified). The strategies become obvious when the error effect that appears in the bias computation is separated from (24). The expression (24) can be rewritten as

\[ -e_N \begin{bmatrix} \tilde{u}(k + 1 - N) \\ \vdots \\ \tilde{u}(k + H_p - N) \end{bmatrix} + e_N \begin{bmatrix} \tilde{u}(k - N) \\ \vdots \\ \tilde{u}(k) \end{bmatrix}, \]  
(27)

where the second term is the (constant) effect that appears in the bias computation (cf. the term \(-s(N)\tilde{u}(k - N)\) in (6)), and the first term is the error that occurs in the predictions when the bias estimate is not applied (cf. the term \(s(N)\tilde{u}(k + j - N)\) in (6)). The expression (27) indicates that modifications introduced into the standard implementation, using the assumption \(s(N + 1) \approx s(N)\), can be made in such a way that the same truncation error effect appears in both terms of (27). In principle, since the past inputs are known and a good estimate of \(e_N\) can be computed during the step response model identification process, a possible approach is to compute and cancel out the first term of (27), instead of using the constant term obtained from the bias estimate.

The above discussions apply directly to models A and C since they are equivalent (see Appendix C), and also to model B if the enhancement in Section 6.2 is implemented.

6.4. A compact and robust standard implementation

This section discusses a compact variant of the standard implementation that has similar computational advantages compared to model B. Moreover, the strategy used in the derivation ensures that cumulative error effects are not introduced into the model. The compact implementation is achieved by modifying the matrix-vector implementation of model A (2). The modifications transfer most of the effects of the third term in (2) (i.e. \(s(N)\tilde{u}(k + j - N)\)) to the second term, \(\sum_{i=j+1}^{N-1} s(i)\Delta\tilde{u}(k + j - i)\). This results in an implementation that has no zeros in its free response matrices and avoids solving (or recovering) all past inputs.

The MIMO model (A.3) is used in the following derivation, where the matrix \(\Psi_{i,j}^A\) is redefined as

\[ \Psi_{i,j}^A = \begin{bmatrix} s_{i,j}(2) & s_{i,j}(3) & \cdots & s_{i,j}(N_j - 2) & s_{i,j}(N_j - 1) \\ s_{i,j}(3) & s_{i,j}(4) & \cdots & s_{i,j}(N_j - 1) & s_{i,j}(N_j) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ s_{i,j}(H_p + 1) & s_{i,j}(H_p + 2) & \cdots & s_{i,j}(N_j) & s_{i,j}(N_j) \end{bmatrix} \]  

where the zeros in \(\Psi_{i,j}^A\) are replaced by \(s_{i,j}(N_j)\). The terms introduced into the prediction model can be extracted from \(\Psi_{i,j}^A\Delta\hat{U}(k)\):

\[ \begin{bmatrix} s_{i,j}(N_j) \\ s_{i,j}(N_j) \\ \vdots \\ s_{i,j}(N_j) \end{bmatrix} \begin{bmatrix} \Delta\tilde{u}(k + H_p - N_j) \\ \Delta\tilde{u}(k + 2 - N_j) \end{bmatrix} \]  
(28)

Rearranging (28), and making an extension to include \(\tilde{u}_j(k + 1 - N_j)\) leads to

\[ \begin{bmatrix} s_{i,j}(N_j) \\ \vdots \\ s_{i,j}(N_j) \end{bmatrix} \begin{bmatrix} \tilde{u}_j(k + 1 - N_j) \\ \tilde{u}_j(k + 2 - N_j) \end{bmatrix} = \begin{bmatrix} \Delta\tilde{u}_j(k + H_p - N_j) \\ \Delta\tilde{u}_j(k + 2 - N_j) \end{bmatrix} \]  
(29)

where \(N_j, \tilde{U}_j(k - N_j)\), and \(Y_{i,j}^A\) are defined in Appendix A.1.

From the above derivation, it is clear that replacing the zeros in \(\Psi_{i,j}^A\) by \(s_{i,j}(N_j)\) is equivalent to extracting the right-side elements of the partitioned matrix and the lower-side elements of the partitioned vector from the \(Y_{i,j}^A\tilde{U}_j(k - N_j)\) term. The remaining elements in the partitioned matrix and vector provide new definitions for \(Y_{i,j}^A\) and \(\tilde{U}_j(k - N_j)\):

\[ Y_{i,j}^A = \begin{bmatrix} s_{i,j}(N_j), s_{i,j}(N_j), \ldots, s_{i,j}(N_j) \end{bmatrix}^T, \]
\[ \tilde{U}_j(k - N_j) = \tilde{u}_j(k + 1 - N_j). \]

This derivation provides a way of canceling out truncation error effects that may be introduced by extending the step response models. In order to verify this claim, consider the simple step response data modification in (23), where the small truncation error is defined as \(e_N = s_{i,j}(N_j) - s_{i,j}(N_j - 1)\). Using this data implies that the error effect

\[ \begin{bmatrix} \Delta\tilde{u}_j(k + H_p - N_j) \\ \Delta\tilde{u}_j(k + 2 - N_j) \end{bmatrix} \]  
(30)
is introduced in the extension (28). This error effect is canceled out by the corresponding effect in the new term $Y^
u_k \hat{U}_j(k - N_j)$:

$$
\begin{bmatrix}
(s_{i,j}(N_j) - e_N)\hat{u}_j(k + 1 - N_j) \\
\vdots \\
(s_{i,j}(N_j) - e_N)\hat{u}_j(k + 1 - N_j)
\end{bmatrix},
$$

where the error effect becomes more obvious when written as

$$
-e_N(\hat{u}_j(k + 2 - N_j) - \Delta \hat{u}_j(k + 2 - N_j)) \\
\vdots \\
-e_N(\hat{u}_j(k + H_P - N_j) - \Delta \hat{u}_j(k + H_P - N_j))
$$

$$
= -e_N \begin{bmatrix}
\hat{u}_j(k + 2 - N_j) \\
\vdots \\
\hat{u}_j(k + H_P - N_j)
\end{bmatrix} + e_N \begin{bmatrix}
e_N \\
\vdots \\
e_N
\end{bmatrix} \begin{bmatrix}
\Delta \hat{u}_j(k + H_P - N_j) \\
\vdots \\
\Delta \hat{u}_j(k + 2 - N_j)
\end{bmatrix}.
$$

The second term cancels out the effect of (30), and the remaining first term is the error effect that will be incurred in model A if the same truncated data is used (cf. (29) or the first term of (27)). Therefore, no cumulative error effects are introduced through the modifications made to model A in order to achieve the compact implementation.

Furthermore, the resulting matrix $\Psi^A_{i,j}$ can be extended by including an extra column using the assumption $s_{i,j}(N_j + 1) \approx s_{i,j}(N_j)$, without introducing any cumulative error effects. This provides a convenient strategy that enables the use of the same past input and disturbance vectors for computing the current output $y(k)$ as well as the $H_P$-step ahead prediction $\hat{y}_f(k + j)$ that involves $\Psi^A_{i,j}$.

6.5. Summary of performance properties

From the discussions in the preceding sections, it is clear that all the different implementations produce similar prediction accuracies if the proposed enhancements are implemented where applicable. The MPC simulations are free from the destabilizing effect shown in Fig. 3 when the residual (26) ignored in model B is implemented, and also when the compact variant in Section 6.4 is implemented. Therefore, the discussions in the remaining sections will be focused on the recursive implementation and the standard implementation (as in model A in Section 2.4.1) and its compact variant (as in Section 6.4). Specifically, the following sections will refer to the prediction models as either recursive, standard, or standard-compact implementations.

7. Performance capabilities in the presence of disturbances

Since the errors (either in the model or state vector) discussed in the previous sections lead to effects that are not properly handled by bias updates, such errors fall in the same category as other unknown disturbances that can not be properly handled by bias updates. A common aspect of the errors is that their effect is propagated through the state vector in the same way. This implies that an error observed in the state vector may be due to the effect of either truncation errors, initialization errors, errors in the measured disturbance history, or an error in the particular implementation. In other words, different errors lead to disturbances in the state.

Therefore, this section uses a simulation example to summarize how the bias updates work in the presence of different disturbances, affecting the plant output, input, and state. In order to clearly show the effect of the simulated disturbances, prediction model implementations that are free from cumulative error effects are used in the simulations.

7.1. Example 3: disturbances on output, input, and state

This example focuses on the performance of step response MPC in the presence of both measured and unknown disturbances, for applications where the use of bias updates is considered. The simple system (25) and the same MPC parameters used in Example 1 are used in this example. The system is initialized at rest with all disturbances, outputs, and input, at zero, and the output reference is set to $r_{y_2} = 0$. Both unknown input and output disturbances and measured disturbances are simulated. The simulation results are shown in Fig. 4. Since different errors lead to disturbances in the state, an error may as well be introduced directly into the state vector. Disturbances in the state are therefore simulated by introducing errors into the state vector, and this produces the results labeled ‘with error’. This case will be discussed after the MPC performance in the presence of output and input disturbances are discussed.

Consider the results labeled ‘without error’ in Fig. 4. The results show the performance of both the recursive and standard implementations when disturbances that affect the plant output and input are simulated at different time instances, and the measured disturbance sequence in the last plot is used. At time $k = 10$, a unit step disturbance (i.e. $w_1 = 1$) starts to act...
on the plant output $y_1$, and it stays active throughout the simulation period. Even though the disturbance is unknown to the controller, $y_1$ is returned to zero after a few time steps. This is simply because this type of disturbance is consistent with the disturbance model incorporated through the bias updates, and therefore detected immediately (see bias plot in Fig. 4). An input step disturbance $\omega = -1$ (unknown to the controller), becomes active from $k = 85$. This disturbance is also handled by the controller, but since this type of disturbance is propagated through the plant, the entire effect is detected after several sampling intervals. Note that the controller is also able to suppress the effect of the measured disturbance sequence, although ramps are simulated in some time intervals. The controller’s response to the changes in measured disturbance is similar to that of an input disturbance, and therefore smooth, but slow, in some intervals.

The simulation sequence described above is repeated for the case where the state of the standard implementation is affected by disturbances. When reconstructing the state in the standard implementation, a step error is introduced into the past disturbance data by using $\Delta d(k - N + 2)$ (instead of $\Delta d(k - N + 1)$) in computing the oldest data point. The error in the state vector becomes significant whenever the measured disturbance signal changes direction, and this leads to the performance results labeled ‘with error’. Note that the state disturbance in this case is due to an implementation error. However, as discussed earlier in this section, the effect illustrated may as well be due to truncation error effects, or some other error effects that are propagated through the state vector. Compared to the input and output disturbances, the state disturbances lead to the worst behavior in this simulation example.

As discussed in Section 5.3, the error in the state vector introduces effects that are not properly handled by the bias updates. Specifically, wrong control moves are computed before suitable bias estimates are computed (cf. Section 5.3 and Example 1). Since the error in this case is recurrent, it is easy to verify the characteristic response sequence that occurs when bias updates are used. In this example, the change in disturbance at $k = 30$ leads to the erroneous control move at $k = 30 + 13 = 43$ followed by a significant change in the bias estimate. Similarly, the disturbance change at $k = 35$ leads to the faulty control move at $k = 35 + 13 = 48$ and the corresponding changes in the output and bias updates.

Since the unknown disturbances discussed in this example are compensated for by the bias update after detection, the adverse effects do not become severe enough to destabilize the plant. However, the MPC performance degrades due to the resultant effect of unknown disturbances and bias updates. Disturbances similar to those discussed in this example are encountered in a more complex industrial case study (presented in the next section). Therefore, this example makes the effects of errors (or unknown disturbances) in the case study more obvious.

8. Industrial case study

This case study verifies the practical importance and benefits of the discussions made in this paper. The control of a subsea compact separation process using a relatively simple embedded MPC framework is discussed. The MPC design and setup is based on Statoil’s MPC software tool presented next.

8.1. SEPTIC MPC scheme and embedded MPC design

The main features of SEPTIC are presented in Strand and Sagli (2003). The intention of this section is to highlight the features that are directly translated into the embedded MPC scheme used in this work. Significant differences are also outlined.

In SEPTIC MPC, limits can be defined for MVs, MV moves, and CVs. However, a quadratic slack variable implementation is used to soften constraints on CVs. Note that an ‘exact penalty’ slack variable implementation is used in (20), where only one slack variable per constraint is needed. In a quadratic or 1-norm formulation, a separate slack variable for every constraint at every CV evaluation point is normally used (see e.g. Maciejowski (2002)), and this may result in a larger QP problem. In addition to weights on control targets, the priority of each control target (including priorities on constraints) can be assigned explicitly in SEPTIC. SEPTIC MPC solves a sequence of steady-state quadratic programs to respect the specified control targets with as many of the high priority control targets as possible, and the steady-state targets are used as references for the dynamic optimization problem. This feature is not implemented in the embedded MPC scheme.

In this work, SEPTIC is used to obtain the step response models and MPC configurations, which serve as a high performance control target specification for the embedded controller. The CV evaluation points are automatically generated by SEPTIC based on information from the step response models, prediction horizon, and MV blocks specified by the user. The priority on each constraint specified in SEPTIC is also considered as a tuning parameter when calculating appropriate weights, $p_h$ and $p_l$, for the embedded MPC’s slack variable implementation.

8.2. Subsea compact separator

A detailed description of the subsea compact separation process can be found in (Hoydal et al., 2013). The process consists of separating a multiphase input flow of liquid (oil/water) and gas at two stages (see Fig. 5). First, a Gas-Liquid Cylindrical Cyclone (GLCC) separates the liquid and gas coarsely, and at the second stage a phase splitter and a de-liquidizer are used for finer separation. The main objective is to control the quality of fluid (i.e. gas volume fraction) in the gas ($\text{Gas}_{\text{out}}$) and liquid
(Liqout) outlets of the separator. It is also necessary to control two pressure variables (P1 and P2) around their operational points and within their safety limits, while respecting the physical limits of three control valves (labeled MV1, MV2, MV3 in Fig. 5). The valves labeled uI and uI+ are controlled by dedicated controllers that provide safety level control for the liquid levels hI and hI+ and ensure that MPC operates on a stable process. The variations in the liquid and gas contents of the inlet flow can be measured, and are considered as two time varying disturbances affecting the process. Due to lack of buffer volumes in the compact separator, the dynamics of the process is much faster compared to most separation techniques, and disturbance effects are much more significant. A sampling frequency of 1Hz or faster is therefore appropriate for high control performance in the presence of challenging inlet flow scenarios such as hydrodynamic slugging.

The process description naturally leads to a constrained multivariable control problem, and MPC is the preferred control method. Moreover, an embedded MPC solution is desirable since the compact separator is to be placed at the sea bed and has fast dynamics due to very small buffer volumes.

8.3. MPC problem setup

The MPC problem setup used in this paper is the same as that used in Kufarol et al. (2014, 2015a,b). The setup includes 4 CVs, each with 10 evaluation points. 1 CV (Liqout) has a high bound, 1 CV (Gasout) has a low bound, and 2 CVs (P1 and P2) have both high and low bounds. There are 6 slack variables (1 slack on each CV bound), 3 MVs (control valves), each with 6 move blocking indices, an upper limit, a lower limit, and a rate of change limit. The evaluation points and move blocks are chosen within a horizon of 80. Two measured process disturbances (DVs) are also present, where a challenging hydrodynamic slugging scenario is simulated by the DV sequences shown in Fig. 6. The above MPC problem description is the same for both the embedded MPC and SEPTIC MPC. In addition, all the controlled variables and constraints have priorities that are implemented explicitly in SEPTIC MPC.

Step response models of lengths shown in Table 1 were obtained using SEPTIC and a nonlinear simulator, where all the models are such that \( s(N_{i,j} + 1) \approx s(N_{i,j}) \). In order to obtain appropriate matrix-vector sizes in the MIMO step response prediction models, simple extensions were made based on \( s(N_{i,j} + 1) \approx s(N_{i,j}) \) and the structure of \( \Psi \) and \( \mathcal{H} \). For the recursive implementation, the largest data length in Table 1 (i.e. 94) is used. For the standard-compact implementation, \( N_j \), defined as the largest \( N_{i,j} \) in Table 1 corresponding to MVj (and similar for DVs), are used. For example, \( N_1 = 93 \) for all models corresponding to MV3, and \( N_2 = 57 \) for all models corresponding to DV1. For the standard implementation, \( N > H_p \) is required. Therefore, the data lengths for the DVs were extended to \( N_{d1} = N_{d2} = 81 \). The choice of data lengths clearly favors the standard-compact implementation, computationally. However, all matrix-vector operations are prepared offline, where the inherent structure of each approach is exploited. This tailoring strategy benefits the recursive approach most, followed by the standard implementation.

8.4. Hardware-in-the-loop simulation setup

The hardware-in-the-loop simulation setup consists of a nonlinear process simulator (i.e. the plant) running on a PC and the embedded MPC application running on an ABB AC500 PM592-ETH1 PLC. The PLC has a 400 MHz Freescale™ G2_LE implementation of the MPC603e microprocessor, 4MB RAM for user program memory, and 4MB integrated user data memory. The PLC communicates with the plant through an OPC server and an Ethernet connection.

Three embedded MPC PLC programs were developed based on the recursive, standard, and standard-compact free response implementations summarized in Section 6.5. The embedded MPC program code is written in C, and it incorporates the QP solver code, which is the same for all implementations. The tailored interior-point algorithm of CVXGEN (Mattingley and Boyd, 2012) was used to obtain high solution accuracies, in double precision. The C code was compiled with gcc 4.7.0, -mcpu=603e, and -01 (which is the highest optimization level possible on the PLC for the CVXGEN code).

8.5. Control performance results

The impact of the free response implementations on separation quality and pressure control in the subsea separator is shown in Table 2. The Mean Square Error values are used as the closed-loop control performance measure. Using the recursive implementation in the embedded MPC scheme produced the same (or slightly better) high-performance control targets as those obtained by SEPTIC MPC on a PC. The standard implementations also achieve similar performance. However, the results become slightly poorer when errors or disturbances affecting the state vector are present.

The bias updates of the embedded MPC controllers labeled without error and with error in Table 2 are shown in Fig. 7, where the standard implementation is used to obtain the results.

---

### Table 1: Number of step response model coefficients (\( N_{i,j} \) or \( d_{i,j} \)) from the \( j \)-th MV or DV to the \( i \)-th CV

<table>
<thead>
<tr>
<th>CVj</th>
<th>MV1</th>
<th>MV2</th>
<th>MV3</th>
<th>DV1</th>
<th>DV2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CV1 (P1)</td>
<td>44</td>
<td>94</td>
<td>93</td>
<td>56</td>
<td>55</td>
</tr>
<tr>
<td>CV2 (P2)</td>
<td>94</td>
<td>94</td>
<td>90</td>
<td>57</td>
<td>56</td>
</tr>
<tr>
<td>CV3 (Liqout)</td>
<td>93</td>
<td>18</td>
<td>81</td>
<td>53</td>
<td>44</td>
</tr>
<tr>
<td>CV4 (Gasout)</td>
<td>93</td>
<td>16</td>
<td>85</td>
<td>51</td>
<td>49</td>
</tr>
</tbody>
</table>
However, the aftermath of the initial response is not smooth for the standard implementation. In both the standard and recursive implementations, slight deviations from the true steady-state values result in a similar initial state of the plant. As discussed in Section 5.3, the slight deviations at start-up only produce an approximation of the true steady-state. This is because the extensions depend on the prediction horizon, implying that the extra computations become less significant when only a few evaluation points are implemented. In this case, 10 evaluation points out of $H_F = 80$ were used. Therefore, compared to the standard implementation, the compact variant produces a similar response sequence that occurs when bias updates are used in the presence of state disturbances (cf. the --- curves). The slugging scenario simulated for the compact separation process leads to complex closed-loop behavior that is difficult to tune when the implementation has the cumulative error effects (28) included in the compact implementation.

The standard implementation, and a “jump” can be seen near the 100 time step, when the slightly erroneous starting history is resolved for all CVs.

### 8.6. Memory usage and computational time

Table 3 shows the memory usage and the computation obtained on the PLC for the compact separator. The results compare the recursive, standard, and standard-compact implementations. Due to the typically large amount of data involved in step response models, the code-generation strategy used is crucial for both the computational time and memory usage. This is illustrated for the standard-compact implementation, where the free response code is also generated by CVXGEN (see the last column of Table 3). The same approach was used in Kuo et al. (2014, 2015b). CVXGEN uses a tailoring approach and parameterization that are suitable for sparse matrices (of relatively small dimensions). This leads to large memory usage and high memory access time when the matrices are dense, as in the case of the standard implementations of the free response. In this work, although CVXGEN is used to generate the QP solver for the MPC setup, the free response computations $Y_f$ are computed separately, and the tailoring strategy used does not depend on the data structures of CVXGEN. This leads to over 50% reduction in data memory usage, and more than $\times 2$ speedup in the computations.

Using the same tailoring strategy for all three implementations (cf. columns 1–3 of Table 3), it is clear that the recursive implementation produces the fastest free response computations and requires the smallest memory size. The recursive implementation is at least $\times 8$ faster than the standard implementations. Data memory usage is similar for all implementations. However, an extra 10KB data memory is used by the standard implementation due to the extra past input and measured disturbance data involved. The standard-compact implementation also uses 60KB extra program memory and produces slightly slower computations compared to the standard implementation.

Note that the computational time is similar for the standard implementations, despite the extra computations incurred due to the extensions (28) included in the compact implementation. This is because the extensions depend on the prediction horizon, implying that the extra computations become less significant when only a few evaluation points are implemented. In this case, 10 evaluation points out of $H_F = 80$ were used. Therefore, compared to the standard implementation, the compact variant...
may be more useful for larger MPC applications where significant data memory is required for storing past inputs and measured disturbances, or when significant computational time is used for recovering the past inputs and disturbances from their respective changes in the past.

Note also that the free response computations may represent the dominating factor in the controller’s total computational time. This is not the case when the CVXGEN solver is used for the compact separator. A total time of over 80ms is obtained using CVXGEN (see e.g. Kuofoalor et al. (2014, 2015b)). However, it is shown in Kuofoalor et al. (2015b) that when the qpOASES (Ferreau et al., 2008) solver is used, a total time below 4ms can be obtained for the same MPC setup. In that case, the time for computing the free response \( Y_f \) becomes the most significant part of the controller’s total computational time.

9. Conclusions

Different implementation approaches for step response prediction models are discussed. It is shown that different step response prediction models, found in the MPC literature, differ only in the way the free response part is calculated. The variants of the standard step response model implementation are shown to be equivalent, and produce the same prediction quality, if the strategies proposed in this paper are used to enhance the models where applicable.

In general, model errors that have cumulative effect lead to the divergence of the output predictions, and the closed-loop system may become unstable. Cumulative error effects may occur in the predictions due to small truncation errors, depending on how the truncated step response data is used to extend the prediction model. It is shown that truncation errors and other unknown disturbances that affect the future prediction computations before appearing in the current output prediction lead to erroneous control moves. Such disturbances are inconsistent with the simple disturbance model (or bias), and the use of bias updates may worsen the effect of the unknown disturbances in some cases.

The results also show that the recursive implementation is the most computationally efficient among the approaches considered in this paper, and it provides the most promising way to implement step response models for large systems on embedded hardware. It is therefore possible to achieve high control performance by using an embedded MPC scheme that has low computational complexity.

Appendix A. MIMO step response prediction models

Due to model linearity, the superposition principle can be used to extend the SISO formulations to MIMO systems. Using (7) and (4) for each CV and MV, consider:

\[
Y(k+1) = \begin{bmatrix} \hat{Y}_1^T(k+1) & \hat{Y}_2^T(k+1) & \ldots & \hat{Y}_{n_{CV}}^T(k+1) \end{bmatrix}^T
\]

\[
Y_f(k+1) = \begin{bmatrix} \hat{Y}_{f1}(k+1) & \hat{Y}_{f2}(k+1) & \ldots & \hat{Y}_{fn_{CV}}(k+1) \end{bmatrix}^T
\]

\[
\Delta U(k) = \begin{bmatrix} \Delta U_1^T(k) & \Delta U_2^T(k) & \ldots & \Delta U_{n_{MV}}^T(k) \end{bmatrix}^T
\]

\[
V(k) = \begin{bmatrix} V_1^T v_1(k) & V_2^T v_2(k) & \ldots & V_{n_{CV}}^T v_{n_{CV}}(k) \end{bmatrix}^T
\]

where the length of \( 1 \) is \( H_p \), \( V(k) \) is the bias variable that represents the effect of unmeasured disturbances, and the number of CVs and MVs in the MIMO system are denoted by \( n_{CV} \) and \( n_{MV} \) respectively.

The approach used to obtain the MIMO models in this paper makes it clear that the prediction models A, B, C, and D differ only in the way the free response part is calculated.

Appendix A.1. MIMO model A

The predictions for MIMO systems can be obtained using the above general vector definitions and the following definitions, specific to the prediction model developed from (3):

\[
\Delta \hat{Y}(k) = \begin{bmatrix} \Delta \hat{Y}_1^T(k) & \Delta \hat{Y}_2^T(k) & \ldots & \Delta \hat{Y}_{n_{MV}}^T(k) \end{bmatrix}^T
\]

\[
\hat{U}(k) = \begin{bmatrix} \hat{U}_1^T(k) & \hat{U}_2^T(k) & \ldots & \hat{U}_{n_{MV}}^T(k) \end{bmatrix}^T
\]

\[
\hat{U}_j(k-N) = \begin{bmatrix} \hat{U}_j^T(k-N_1) & \hat{U}_j^T(k-N_2) & \ldots & \hat{U}_j^T(k-N_{n_{MV}}) \end{bmatrix}^T
\]

\[
\hat{U}_j(k-N) = \begin{bmatrix} \hat{U}_j(k-N_1) & \hat{U}_j(k-N_2) & \ldots & \hat{U}_j(k-N_{n_{MV}}) \end{bmatrix}^T
\]

Note that \( N \) is not assumed to be of the same length for all SISO models. For simplicity, the number of samples \( N_j \) is defined as the largest \( N \) of only the SISO step response models corresponding to MV number \( j \). The MIMO prediction model takes the form:

\[
Y^A(k+1) = \Theta A \Delta U(k) + \Psi A \Delta \hat{Y}(k) + Y^A \hat{U}(k-N) + D(k) + V(k)
\]

\[
\Psi_A = \begin{bmatrix} \Psi_{A,1}^A & \ldots & \Psi_{A,n_{MV}}^A \\
\vdots & \ddots & \vdots \\
\Psi_{A,n_{CV}}^A & \ldots & \Psi_{A,n_{MV}}^A \end{bmatrix}
\]

\[
\Psi_{A,i,j} = \begin{bmatrix} s_{i,j}(2) & s_{i,j}(3) & \ldots & s_{i,j}(N_j-2) & s_{i,j}(N_j-1) \\
s_{i,j}(3) & s_{i,j}(4) & \ldots & s_{i,j}(N_j-1) & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
s_{i,j}(H_p+1) & s_{i,j}(H_p+2) & \ldots & 0 & 0 \end{bmatrix}
\]

The dimension of \( \Psi_A \) is \( n_{CV} (H_p - H_w) \times (n_{MV} \cdot H_p) \) and that of \( \Psi_{i,j}^A \) is \( (H_p - H_w + 1) \times N_j - 2 \). \( \Theta \) is an \( (n_{CV} \cdot H_p) \times (n_{MV} \cdot H_p) \) dimensional matrix of the same form as \( \Psi_A \), and \( \Theta_{i,j} \) is defined in (9). \( Y^A \) is of dimension \( (n_{CV} \cdot H_p) \times (n_{MV} \cdot H_p) \) and also has the same form as \( \Psi_A \), whereas \( \Psi_{i,j} = s_{i,j}(N_j) I_{H_p} \). The measured disturbance term is

\[
D(k) := D_{dA} \Delta D(k) + Y_{dA} \Delta D(k) + \Theta_d \Delta D(k),
\]

where the matrices and vectors involved, with the exception of \( \Theta_d \) and \( \Delta D(k) \), are defined in a similar way as the corresponding components for the MV–CV models stated above. The matrix \( \Theta_d \) of size \( (n_{CV} \cdot H_p) \times n_{CV} \) can also be thought of as similar to \( \Theta \) for an MV, but includes only changes at the current time, i.e.

\[
\Delta D(k) = \begin{bmatrix} \Delta d_1(k) & \Delta d_2(k) & \ldots & \Delta d_{n_{CV}}(k) \end{bmatrix}^T
\]

\[
\Theta_{d,i,j} = \begin{bmatrix} s_{d,i,j}(1) & s_{d,i,j}(2) & \ldots & s_{d,i,j}(H_p) \end{bmatrix}^T
\]
The coefficients that describe the step response from DV \( j \) to CV \( i \) are denoted by \( s_{ij}(\cdot) \), and \( n_{DV} \) is the number of MVs.

Further stacking of variables leads to the compact form:

\[
Y^A(k+1) = \Theta \Delta U(k) + Y^A_f(k+1),
\]

\[
Y^B_f(k+1) = \Theta_d \Delta D(k) + \tilde{\Psi}^A \Delta \tilde{U}(k) + \tilde{\Psi}^A \tilde{U}(k-N) + V(k),
\]

where

\[
\tilde{U}(k-N) = \begin{bmatrix} \hat{U}^T(k-N), \hat{D}^T(k-N) \end{bmatrix}^T
\]

\[
\Delta \tilde{U}(k) = [\Delta \hat{U}^T(k), \Delta \hat{D}^T(k)]^T
\]

\[
\tilde{U}(k-N) = \begin{bmatrix} \hat{U}^T(k-N), \hat{D}^T(k-N) \end{bmatrix}^T
\]

\[
\tilde{\Psi}^A = \begin{bmatrix} \Psi^A, \Psi_d \end{bmatrix}
\]

\[
\tilde{Y}^A = [Y^A, Y_d]
\]

The superscript \( A \) is used in (A.3) to indicate that \( Y^A(k+1) \) and \( Y^B_f(k+1) \) are variants of \( Y(k+1) \) and \( Y_f(k+1) \) respectively.

**Appendix A.2. MIMO model B**

Another MIMO model can be developed based on (11):

\[
Y^B(k+1) = \Theta \Delta U(k) + Y^B_f(k+1),
\]

\[
Y^B_f(k+1) = \Theta_d \Delta D(k) + \tilde{\Psi}^B \Delta \tilde{U}(k) + \tilde{\Psi}^B \tilde{U}(k-N) + V(k),
\]

where the matrices involved follow similar definitions used in variant A, with the exception of

\[
\tilde{\Psi}^B_{i,j} = \begin{bmatrix} \tilde{s}_{ij}(1) & \tilde{s}_{ij}(2) & \tilde{s}_{ij}(H_p) \end{bmatrix}^T,
\]

\[
\tilde{\Psi}^B_{i,j} = \begin{bmatrix} \tilde{s}_{ij}(2) & \tilde{s}_{ij}(3) & \ldots & \tilde{s}_{ij}(N_j-1) & \tilde{s}_{ij}(N_j) \\
\tilde{s}_{ij}(3) & \tilde{s}_{ij}(4) & \ldots & \tilde{s}_{ij}(N_j) & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\tilde{s}_{ij} & \tilde{s}_{ij}(H_p+1) & \tilde{s}_{ij}(H_p+2) & \ldots & 0 & 0 \\
\end{bmatrix}
\]

Note that \( \Delta \tilde{U}_j(k) \) in \( \Delta \tilde{U}(k) \) will become

\[
\Delta \tilde{U}_j(k) = \begin{bmatrix} \Delta \tilde{U}_j(k-1) & \Delta \tilde{U}_j(k-2) & \ldots & \Delta \tilde{U}_j(k-N_j+1) \end{bmatrix}^T,
\]

\[
\tilde{U}(k-1) = \begin{bmatrix} \tilde{u}_1(k-1) & \ldots & \tilde{u}_{ny}(k-1) & \ldots & \tilde{u}_{ny}(k-1) \end{bmatrix}^T,
\]

and a corresponding definition applies to \( \Delta \tilde{D}_j(k) \). The step response coefficient \( \tilde{s}_{ij}(\cdot) \) represents either \( s_{ij}(\cdot) \) or \( \tilde{s}_{ij}(\cdot) \) in the corresponding order defined in \( \tilde{U}(k-1) \).

**Appendix A.3. MIMO model C**

A convenient MIMO model can be obtained when the output measurement is treated as a separate term in the state equation (14), leading to

\[
Y_m(k) = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix} Y_m(k) + Y_m(k) + Y_{m\text{CV}}(k)
\]

\[
\hat{X}_m(k) = \begin{bmatrix} \Delta \hat{X}_m(k-1) & \Delta \hat{X}_m(k-2) & \ldots & \Delta \hat{X}_m(k-N_m+1) \end{bmatrix}^T,
\]

\[
\hat{X}_{m\text{CV}}(k) = \begin{bmatrix} \Delta \hat{X}_{m\text{CV}}(k-1) & \Delta \hat{X}_{m\text{CV}}(k-2) & \ldots & \Delta \hat{X}_{m\text{CV}}(k-N_{m\text{CV}}+1) \end{bmatrix}^T,
\]

\[
\hat{X}(k) = \begin{bmatrix} \hat{X}_m(k) & \hat{X}_{m\text{CV}}(k) & \hat{X}_{d1}(k) & \ldots & \hat{X}_{d_{n\text{MV}}}(k) \end{bmatrix}^T,
\]

\[
\hat{X}_m \text{ is of size } (n_{\text{CV}} \cdot H_p) \times (n_{\text{MV}} + n_{n\text{MV}})(N-1) \text{ and of form similar to (A.1), and } \hat{s}_{ij} \text{ is the same as (15) without the first column. The MIMO model becomes}
\]

\[
Y^C(k+1) = \Theta \Delta U(k) + Y^C_f(k+1),
\]

\[
Y^C_f(k+1) = \Theta_d \Delta D(k) + Y_m(k) + \omega_f \hat{X}(k).
\]

**Appendix A.4. MIMO model D**

The recursive implementation can be extended to MIMO systems by rewriting (19) using block matrices/vectors. The result is

\[
Y(k) = \alpha Y(k-1) + \beta \tilde{D}(k-1) + \beta_d \tilde{D}(k-1)
\]

\[
Y^D_f(k+1) = \Theta \Delta U(k) + Y^D_f(k+1),
\]

\[
Y^D_f(k+1) = \Theta_d \Delta D(k) + \alpha_p Y(k) + V(k)
\]

where

\[
\alpha = \text{blkdiag}(\alpha_1, \alpha_2, \ldots, \alpha_{n_{\text{CV}}})
\]

\[
\alpha_p = \text{blkdiag}(\alpha_{p1}, \alpha_{p2}, \ldots, \alpha_{p_{n_{\text{CV}}}})
\]

\[
\gamma = \text{blkdiag}(\gamma_1, \gamma_2, \ldots, \gamma_{n_{\text{CV}}})
\]

\[
\Delta \tilde{U}(k-1) = [\Delta \tilde{U}_1(k-1) \ \Delta \tilde{U}_2(k-1) \ \ldots \ \Delta \tilde{U}_{n_{\text{CV}}}(k-1)]^T
\]

\[
\Delta \tilde{D}(k-1) = [\Delta \tilde{D}_1(k-1) \ \Delta \tilde{D}_2(k-1) \ \ldots \ \Delta \tilde{D}_{n_{\text{CV}}}(k-1)]^T
\]

\[
Y(k) = \begin{bmatrix} Y_1^T(k-1) & Y_2^T(k-1) & \ldots & Y_{n_{\text{CV}}}^T(k-1) \end{bmatrix}^T
\]

\[
Y(k) = \begin{bmatrix} Y_1^T(k) & Y_2^T(k) & \ldots & Y_{n_{\text{CV}}}^T(k) \end{bmatrix}^T
\]

Both \( \beta \) and \( \beta_d \) are of the same structure as (A.1), composed of \( \beta \) and their sizes are \( (n_{\text{CV}} \cdot N) \times n_{\text{MV}} \) and \( (n_{\text{CV}} \cdot N) \times n_{\text{DV}} \), respectively.

**Appendix B. Derivation of the residual ignored in model B**

Model B (11) is derived in (Maciejowski, 2002, §4.1.3). The first and last but one steps are repeated here (for a SISO system) in order to facilitate direct comparison:

\[
y(k+j|k) = \sum_{i=1}^{N} h(i)u(k+j-i)
\]

\[
= \sum_{i=j+1}^{N} h(i)\tilde{u}(k+j-i) + \sum_{i=1}^{j} h(i)\tilde{u}(k-1)
\]

\[
+ (h(1)\Delta u(k+j-1))
\]

\[
+ [h(1)+h(2)]\Delta u(k+j-2) + \ldots
\]

\[
+ [h(1)+h(2)+\ldots+h(j)]\Delta u(k-j)],
\]

where the impulse response sequence \{\( h(0), \ldots, h(N) \)\} is used, and the inputs and impulse response coefficients are adapted to the notations of this paper.

The second term and last group of terms (in parentheses) in (B.2) lead to the second and last terms of (11), respectively. However, the remaining first term of (11) has a \"residual\" that becomes obvious when the first term is written out for each \( j \). For simplicity and clarity of presentation, consider \( k = 0 \), \( j = 1, \ldots, H_p = 3 \), \( N = 4 \), \( s_i = s(i) \), and \( h_i = h(i) \). The first term of (11) can be written out as:

\[
 j = 1 : s_2 \Delta \tilde{u}(k-1) + s_3 \Delta \tilde{u}(k-2) + s_4 \Delta \tilde{u}(k-3)
\]

\[
 j = 2 : s_3 \Delta \tilde{u}(k-1) + s_4 \Delta \tilde{u}(k-2)
\]

\[
 j = 3 : s_4 \Delta \tilde{u}(k-1)
\]
The first term of (B.2) also becomes:

\[ j = 1 : h_2 \hat{u}(-1) + h_3 \hat{u}(-2) + h_4 \hat{u}(-3) \quad (B.4a) \]
\[ j = 2 : h_3 \hat{u}(-1) + h_4 \hat{u}(-2) \quad (B.4b) \]
\[ j = 3 : h_4 \hat{u}(-1) \quad (B.4c) \]

Recall that \( h_j = s_j - s_{j-1} \). By direct substitution, and using \( \Delta u(j) = u(j) - u(j-1) \), (B.4) becomes

\[ 1 : s_2 \Delta \hat{u}(-1) + s_3 \Delta \hat{u}(-2) + s_4 \Delta \hat{u}(-3) + s_1 \hat{u}(-1) \]
\[ 2 : s_3 \Delta \hat{u}(-1) + s_4 \Delta \hat{u}(-2) + s_4 \hat{u}(-3) - s_2 \hat{u}(-1) \]
\[ 3 : s_4 \Delta \hat{u}(-1) + s_4 \hat{u}(-2) - s_3 \hat{u}(-1) \]

where the last two terms (in red) of each line represent the residual incurred in (B.4) compared to (B.3) at each stage \( j \) in the prediction horizon. Note that, using \( \Delta u(j) = u(j) - u(j-1) \), the residual terms can be rewritten as

\[ y^B_{res}(k+1) = s_4 \hat{u}(-3) - s_3 \Delta \hat{u}(-3) - s_1 \hat{u}(-1) \]
\[ y^B_{res}(k+2) = s_4 \hat{u}(-2) - s_2 \Delta \hat{u}(-2) - s_2 \hat{u}(-1) \]
\[ y^B_{res}(k+3) = s_4 \hat{u}(-1) - s_3 \Delta \hat{u}(-1) - s_3 \hat{u}(-1) \]

From the above result, the entire residual of (11) becomes

\[ y^B_{res}(k+j) = s(N)\hat{u}(k+j-N) - s(N)\hat{u}(k-1). \quad (B.5) \]

It is now evident that adding (B.5) to (11) makes (11) equal to (10), and hence model B the same as model A.

Appendix C. Robustness of model C compared to model A

Model C can be obtained by rewriting (6) as

\[ \hat{y}_f(k+j|k) = y_m(k) + \sum_{i=1}^{N} \left( s(i+j) - s(i) \right) \Delta \hat{u}(k - i) \]
\[ + (s(N) - s(N+j)) \Delta \hat{u}(k - N) \]
\[ + s(N)\hat{u}(k+j-N) - s(N)\hat{u}(k-1). \quad (C.1) \]

It is obvious that the last two lines are not implemented in model C (cf. (12)). Using the assumption that \( s(N+j) - s(N) \approx 0 \), for \( j \geq 1 \), implies that the second line can be ignored since any small truncations errors introduced will be fairly constant over the prediction horizon. On the other hand, the last term cannot be assumed to be always zero or constant for \( j = 1, \ldots, H_p \), and should therefore be implemented.

However, the first term is prone to cumulative error effects due to the extension made through the use of \( s(N+j) - s(N) \approx 0 \). Therefore the equivalent canceling term \( -\sum_{i=1}^{N} s(N)\hat{u}(k+i) \) can be included to (C.1), which becomes equal to model A. Furthermore, it is straightforward to show that \( s(N)\hat{u}(k+j-N) - s(N)\hat{u}(k-N) = \sum_{i=1}^{N} s(N)\Delta \hat{u}(k+i-N) \). Therefore the terms in the last line of (C.1) can be canceled out instead. This leads to model C:

\[ \hat{y}_f(k+j|k) = y_m(k) + \sum_{i=1}^{N} \left( s(i+j) - s(i) \right) \Delta \hat{u}(k - i), \]

which is as robust as model A, considering truncation error effects.

Acknowledgment

This work was supported by the Research Council of Norway and Statoil through the PETROMAKS project 215684. We thank G. O. Eikrem, M. Fredriksen, and S. Strand at Statoil, for useful discussions on step response models and implementation aspects of SEPTIC. We also thank Statoil for making SEPTIC MPC tools and models available for this study.

References


16