On computing frequency-responses of periodic systems

A. Varga

***Abstract*— We address the efficient and numerically reliable computation of frequency responses of discrete-time periodic systems by using structure exploiting algorithms.**

I. INTRODUCTION

We consider periodic time-varying descriptor systems of the form

$$\begin{array}{rcl} E_k x(k+1) &=& A_k x(k) + B_k u(k) \\ y(k) &=& C_k x(k) + D_k u(k) \end{array} \qquad (1)$$

where the matrices $E_k \in \mathrm{R}^{\mu_k \times n_{k+1}}$, $A_k \in \mathrm{R}^{\mu_k \times n_k}$, $B_k \in \mathrm{R}^{\mu_k \times m_k}$, $C_k \in \mathrm{R}^{p_k \times n_k}$, $D_k \in \mathrm{R}^{p_k \times m_k}$ are periodic with period $N \geq 1$, and the dimensions fulfil the condition $\nu = \sum_{k=1}^{N} \mu_k = \sum_{k=1}^{N} n_k$.

To define the frequency response matrix of the periodic system (1), we define first the *transfer-function matrix* (TFM) corresponding to the associated *stacked lifted representation* of [1], which uses the input-state-output behavior of the system over time intervals of length N, rather then 1. For a given sampling time k, the corresponding M-dimensional input vector, P-dimensional output vector and ν-dimensional state vector are

$$\begin{array}{rcl} u_k^L(h) &=& [u^T(k+hN) \cdots u^T(k+hN+N-1)]^T, \\ y_k^L(h) &=& [y^T(k+hN) \cdots y^T(k+hN+N-1)]^T, \\ x_k^L(h) &=& [x^T(k+hN) \cdots x^T(k+hN+N-1)]^T. \end{array}$$

where $M = \sum_{k=1}^{N} m_k$ and $P = \sum_{k=1}^{N} p_k$. The corresponding *stacked* lifted system can be represented by a time-invariant descriptor system of the form

$$\begin{array}{rcl} L_k x_k^L(h+1) &=& F_k x_k^L(h) + G_k u_k^L(h) \\ y_k^L(h) &=& H_k x_k^L(h) + J_k u_k^L(h) \end{array} \qquad (2)$$

where

$$F_k - zL_k = \begin{bmatrix} -zE_{k+N-1} & O & \cdots & O & A_{k+N-1} \\ A_k & -E_k & O & \cdots & O \\ O & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -E_{k+N-3} & O \\ O & & \ddots & A_{k+N-2} & -E_{k+N-2} \end{bmatrix} \qquad (3)$$

$$G_k = \begin{bmatrix} O & O & \cdots & O & B_{k+N-1} \\ B_k & O & \cdots & O & O \\ \vdots & \ddots & & \ddots & \vdots \\ O & O & B_{k+N-2} & & O \end{bmatrix}$$

A. Varga is with the German Aerospace Center, DLR - Oberpfaffenhofen, Institute of Robotics and Mechatronics, D-82234 Wessling, Germany. E-mail: andras.varga@dlr.de

$$H_k = \mathrm{diag}\,(C_k, \ldots, C_{k+N-1}),$$
$$J_k = \mathrm{diag}\,(D_k, \ldots, D_{k+N-1})$$

Assuming the square pencil (3) is regular (i.e., $\det(F_k - zL_k) \not\equiv 0$), the TFM of the lifted system is

$$W_k(z) = H_k(zL_k - F_k)^{-1} G_k + J_k \qquad (4)$$

and depends on the sampling time k.

For standard periodic systems (i.e., $E_k = I_{n_{k+1}}$), we can use another lifted time-invariant representation introduced in [2], which corresponds to the same lifted input and output vectors, but to an n_k-dimensional state vector defined as

$$\widehat{x}_k(h) := x(k+hN).$$

The transition matrix of the system (1) for $E_k = I_{n_{k+1}}$ is defined by the $n_j \times n_i$ matrix $\Phi_A(j,i) = A_{j-1}A_{j-2}\cdots A_i$, where $\Phi_A(i,i) := I_{n_i}$. The *standard* lifted system has the form

$$\begin{array}{rcl} \widehat{x}_k(h+1) &=& \widehat{F}_k \widehat{x}_k(h) + \widehat{G}_k u_k^L(h) \\ y_k^L(h) &=& \widehat{H}_k \widehat{x}_k(h) + \widehat{J}_k u_k^L(h) \end{array} \qquad (5)$$

where

$$\begin{array}{rcl} \widehat{F}_k &=& \Phi_A(k+N,k) \\ \widehat{G}_k &=& [\ \Phi_A(k+N,k+1)B_k\ \cdots B_{k+N-1}\ ] \end{array}$$

$$\widehat{H}_k = \begin{bmatrix} C_k \\ C_{k+1}\Phi_A(k+1,k) \\ \vdots \\ C_{k+N-1}\Phi_A(k+N-1,k) \end{bmatrix}$$

$$\widehat{J}_k = \begin{bmatrix} D_k & 0 & \cdots & 0 \\ \widehat{J}_{k,2,1} & D_{k+1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \widehat{J}_{k,N,1} & \widehat{J}_{k,N,2} & \cdots & D_{k+N-1} \end{bmatrix}$$

with $\widehat{J}_{k,i,j} = C_{k+i-1}\Phi_A(k+i-1,k+j)B_{k+j-1}$, for $i = 2, ..., K$, $j = 1, 2, \ldots N-1$, and $i > j$.

The associated TFM $\widehat{W}_k(z)$ is

$$\widehat{W}_k(z) = \widehat{H}_k(zI_{n_k} - \widehat{F}_k)^{-1}\widehat{G}_k + \widehat{J}_k \qquad (6)$$

Since for $E_k = I_{n_{k+1}}$, $W_k(z)$ defined in (4) and $\widehat{W}_k(z)$ defined in (6) coincide, $W_k(z)$ appears to be the natural generalization of the TFM for a standard periodic system to the descriptor case, covering also the most general representation of descriptor periodic systems with time-varying dimensions.

For a given $\theta \in [\,0,\, 2\pi\,]$, the frequency-response matrix $W_k(e^{j\theta})$ is given by

$$W_k(e^{j\theta}) = H_k(e^{j\theta}L_k - F_k)^{-1}G_k + J_k$$

If T_s is the sampling period of the system, then θ can be expressed as $\theta = \omega T_s$, where ω is the frequency.

Obviously $W_{k+N}(z) = W_k(z)$ and the TFMs at two successive values of k are related by the relation [3]

$$W_{k+1}(z) = \begin{bmatrix} 0 & I_{P-p_k} \\ zI_{p_k} & 0 \end{bmatrix} W_k(z) \begin{bmatrix} 0 & z^{-1}I_{m_k} \\ I_{M-m_k} & 0 \end{bmatrix} \quad (7)$$

Therefore, for the simplicity of notation we drop in what follows the dependence of index k, assuming $k = 1$. The values of the frequency response for values of $k > 1$ can be computed by a trivial permutation of data, or using the above relations in the case when the frequency-response must be computed simultaneously for several values of k.

The computation of the frequency-response usually requires the evaluation of $W(\lambda_i)$ at a large number of values $\lambda_i = e^{j\theta_i}$, $i = 1, \ldots, N_f$. This involves for each λ_i the solution of the complex linear equation $(\lambda_i L - F)X = G$ and computing $W(\lambda_i) = HX + J$. Note that all intervening matrices are usually large but highly structured. Without exploiting their structure, the computational effort can be prohibitive.

In this paper we discuss several structure exploiting approaches to compute efficiently the frequency-response. Since the numerical properties of the underlying algorithms are determined by the properties of the employed linear system solvers, all methods we discuss appear to be essentially equivalent. Therefore, to compare different methods, we provide only estimates of the necessary computational effort and additional storage.

The necessary computational costs are expressed in terms of the performed *floating-point operations* or *flops* (1 *flop* = 1 multiplication or 1 addition). For each method, the computational effort N_{op} can be expressed as a sum of two terms

$$N_{op} = N_{op,0} + N_f N_{op,f}$$

where $N_{op,0}$ is the cost of any preliminary preprocessing of problem data (performed usually only once), while $N_{op,f}$ is the cost to evaluate $W(\lambda_i)$ for a single frequency value. The purpose of the preliminary preprocessing is to globally reduce the computational cost compared to the brute force approach when ignoring the underlying structure in solving $(\lambda_i L - F)X = G$.

Note. To obtain simpler expressions for the operation counts, we will make evaluations only for systems with constant dimensions. For systems with time-varying dimensions, the resulting figures represent upper bounds in terms of the maximum dimensions $n = \max_i\{n_i, \mu_i\}$, $m = \max_i\{m_i\}$, $p = \max_i\{p_i\}$.

II. Methods based on solving BABD systems

To evaluate $W(\lambda_i)$, the main computation is to solve the linear system $RX = G$, where $R = \lambda_i L - F$. This is a potentially large order linear system with a so-called *bordered almost block diagonal* (BABD) structured coefficient matrix of the form

$$R = \begin{bmatrix} R_{11} & & & R_{1,N} \\ R_{21} & R_{22} & & \\ & \ddots & \ddots & \\ & & R_{N,N-1} & R_{N,N} \end{bmatrix}$$

where $R_{11} = \lambda_i E_1$, $R_{kk} = E_k$ for $k = 2, \ldots, N$, $R_{i+1,i} = -\widehat{A}_i$ for $i = 1, \ldots, N-1$, $R_{1,N} = -A_N$. The right-hand side has the cyclic structure

$$G = \begin{bmatrix} & & & G_{1N} \\ G_{21} & & & \\ & \ddots & & \\ & & G_{N,N-1} & \end{bmatrix}$$

where $G_{i+1,i} = B_i$, for $i = 1, \ldots, N-1$ and $G_{1,N} = B_N$. The solution X, of the same size as G, is obtained in a $N \times N$ block partitioned form with the $\mu_i \times m_i$ matrix X_{ij} as its generic block element for $i, j = 1, \ldots, N$.

BABD systems appear, for example, when solving two-point boundary value problems for ordinary differential equations with nonseparable constraint using finite-difference methods [4], [5]. There is a rich literature on solving BABD and ABD systems (an ABD is an BADB with $R_{1,N} = 0$), and many methods have been proposed to address the efficient numerical solution of such systems [5]. Among the existing methods, we mention just a few, like applying band linear solvers, transforming BABD into larger order ABD systems and applying efficient ABD solvers, block LU and QR-factorization base methods.

To solve the BABD system of equations $RX = G$ we describe the structured *Gaussian elimination with partial pivoting* (GEPP). Recall that the standard GEPP method to solve the linear equation $RX = G$ has two main steps [6]. First, the LU factorization of R is computed by using partial (row) pivoting, to obtain $PR = LU$, where P is a row permutation matrix, L is a unit lower triangular matrix and U is an upper triangular matrix. Then, the solution is determined by using forward substitution to compute $Y = L^{-1}(PG)$ and back substitution to compute $X = U^{-1}Y$.

For the particular structure of R above, the resulting U has an upper triangular block-structured form

$$U = \begin{bmatrix} U_{11} & U_{12} & & & U_{1N} \\ & U_{22} & U_{23} & & U_{2N} \\ & & \ddots & \ddots & \\ & & & \ddots & U_{N-1,N} \\ & & & & U_{N,N} \end{bmatrix} \quad (8)$$

with nonzero blocks only on the main block-diagonal, first block-supradiagonal and last block-column. Regarding L, nothing can be said in general about its bandwidth, but it is known (see [6]) that each column of the lower triangular part contains at most $2n$ nonzero elements.

As a consequence of the cyclic block-structure of G above, $Y = L^{-1}(PG)$ has a special almost lower triangular block-structure with a nonzero last block-column

$$Y = \begin{bmatrix} Y_{11} & & & Y_{1,N} \\ Y_{21} & Y_{22} & & Y_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{N,1} & Y_{N,2} & \cdots & Y_{N,N} \end{bmatrix} \tag{9}$$

However, the resulting $X = U^{-1}Y$ is generally full, although to compute it the block-structure of U can be efficiently exploited when computing it using block back substitution. For the efficient evaluation of the matrix expression $HX + J$, the block-diagonal structure of the matrices H and J can be exploited.

To compute the block-LU factorization of R the following algorithm can be used:

Algorithm 1: *Block-LU factorization* $PR = LU$.

for $i = 1, \ldots, N-1$

Compute the LU factorization $P_i \begin{bmatrix} R_{ii} \\ R_{i+1,i} \end{bmatrix} = L_i \begin{bmatrix} U_{ii} \\ O \end{bmatrix}$

Compute $\left[\begin{array}{c|c} U_{i,i+1} & U_{i,N} \\ R_{i+1,i+1} & R_{i+1,N} \end{array}\right] := L_i^{-1} P_i \left[\begin{array}{c|c} O & R_{i,N} \\ R_{i+1,i+1} & R_{i+1,N} \end{array}\right]$

end

Compute the LU factorization $P_N R_{N,N} = L_N U_{N,N}$.

The main computations in **Algorithm 1** are the $N-1$ successive LU decompositions of $(\mu_i+\mu_{i+1}) \times n_i$ matrices and the application of n_i elementary transformations to $(\mu_i+\mu_{i+1}) \times (n_{i+1}+n_N)$ matrices. Therefore, for constant dimensions this algorithm performs about $5n^3/3$ flops to compute the LU-decomposition of a $2n \times n$ matrix and additionally, $6n^3$ flops to apply n elementary transformations to a $2n \times 2n$ matrix [6]. Thus, **Algorithm 1** performs about $\max\{(N-1)\frac{23}{3}n^3, \frac{2}{3}n^3\}$ *flops*. Additional $2(N-1)n^2$ storage locations are necessary to store the supradiagonal blocks of U and the blocks in the last block-columns and $2Nn^2$ storage locations are necessary to store L_i. The upper triangular diagonal blocks U_{ii} of U can be stored in the upper diagonal part of L_i. In this way, the L matrix of the LU decomposition is stored implicitly in the lower triangular matrices L_i.

To compute $Y = L^{-1}(PG)$ in (9) (only the non-zero elements), the following forward substitution algorithm can be employed.

Algorithm 2: *Block-forward substitution to solve LY=PG.*

$Y = G$

for $i = 1, \ldots, N-1$

Compute $\left[\begin{array}{ccc|c} Y_{i,1} & \cdots & Y_{i,i} & Y_{i,N} \\ Y_{i+1,1} & \cdots & Y_{i+1,i} & Y_{i+1,N} \end{array}\right] :=$

$$L_i^{-1} P_i \left[\begin{array}{ccc|c} Y_{i,1} & \cdots & Y_{i,i-1} & 0 & Y_{i,N} \\ 0 & \cdots & 0 & Y_{i+1,i} & 0 \end{array}\right]$$

end

Compute $Y_{N,N} := L_N^{-1} P_N Y_{N,N}$.

Algorithm 2 basically applies the performed elementary transformations and permutations to the right hand side G. The computational effort for constant dimensions is $\max\{\frac{3}{2}n^2m(N-1)(N+2), n^2m\}$ *flops*.

To compute $X = U^{-1}Y$ with U of the form (8) and Y of the form (9), we assume that both X and Y are partitioned row-wise compatibly with the column structure of U in N block-rows X_i and Y_i, $i = 1, \ldots, N$, respectively. The following back substitution algorithm can be employed to compute X.

Algorithm 3: *Block-back substitution to solve* $UX = Y$.

Solve $U_{NN} X_N = Y_N$

If $N > 1$, then solve $U_{N-1,N-1} X_{N-1} = Y_{N-1} - U_{N-1,N} X_N$.

for $i = N-2, \ldots, 1$

Solve $U_{ii} X_i = Y_i - U_{i,i+1} X_{i+1} - U_{i,N} X_N$

end

The number of operations for **Algorithms 3** is $\max\{3n^2m(N-1)N, n^2m\}$ *flops*.

When evaluating $W(\lambda_i)$ block-wise as

$$W(\lambda_i) = \begin{bmatrix} W_{11} & \cdots & W_{1N} \\ \vdots & \ddots & \vdots \\ W_{N1} & \cdots & W_{NN} \end{bmatrix}$$

where the (i,j)-th block is $p_i \times m_j$, it is possible to avoid forming the matrices G, H and J. For example, using **Algorithms 1-3**, we can solve successively $R\overline{X}_j = \overline{G}_j$, for $j = 1, \ldots, N$, where $\overline{G}_j$ is formed from the j-th block-column of G. Let X_{ij} be the i-th block-row of $\overline{X}_j$ of size $n_i \times m_i$. Then, the multiplication $H\overline{X}_j + \overline{D}_j$ amounts to compute the products $W_{ij} = C_i X_{ij}$, $i = 1, \ldots, N$, $i \neq j$, and $W_{jj} = C_j X_{jj} + D_j$. The number of operations to evaluate all X_{ij}, assuming constant dimensions, is $2pmnN^2$ *flops*.

To determine $W(\lambda_i)$ the total number of operations is (assuming $N > 1$)

$$N_{op,f} = \left(\frac{23}{3}n^3 - \frac{3}{2}n^2m\right)N + \left(\frac{9}{2}n^2m + 2pmn\right)N^2 \tag{10}$$

The necessary additional storage is not larger than

$$N_{st} = \max\{4n^2N, nmN\}$$

provided the column oriented computation of $W(\lambda_i)$ is employed.

Interestingly, it appears that there exists no general purpose numerically stable method to solve BABDs systems [5]. Moreover, a special class of these systems represents a notorious example for the failure of Gaussian elimination with partial pivoting [7] due to excessive growth factors. Therefore, as an alternative to the structured GEPP, the structured orthogonal QR-decomposition has been suggested in [5]. Having the QR-decomposition $QU = R$, where Q is orthogonal and U is upper triangular, the solution can be computed as $Y = Q^T G$ and $X = U^{-1}Y$.

The resulting Q has an upper block-Hessenberg structure, with a lower bandwidth of $2n$. We can store Q also in a factored form, so to compute Y, there is no need to form explicitly Q. The number of operations to compute Y is roughly twice as much as in the case when employing the GEPP to compute Y using the LU decomposition of R.

III. Methods for standard periodic systems

In this section we discuss methods for standard periodic systems where $E_k = I_{n_{k+1}}$.

A. Hessenberg method

The matrix $R = \lambda_i L - F$ is generally in a block-Hessenberg form, with a lower bandwidth of at most $2n$. By reducing the N-periodic matrix A_k to an *extended periodic Hessenberg form* (EPHF) (a generalization of the periodic Hessenberg form introduced in [8]), we can reduce the lower bandwidth to $n+1$. For convenience we assume $\underline{n} = \min_k\{n_k\} = n_N$. According to [9], given the matrices $A_k \in \mathrm{R}^{n_{k+1}\times n_k}$, $k = 1,\ldots,N$, with $n_{N+1} = n_1$ there exist orthogonal matrices $Q_k \in \mathrm{R}^{n_k\times n_k}$, $k = 1,\ldots,N$, $Q_{N+1} := Q_1$, such that the transformed matrices

$$\widetilde{A}_k := Q_{k+1}^T A_k Q_k = \begin{bmatrix} \widetilde{A}_{k,11} & \widetilde{A}_{k,12} \\ 0 & \widetilde{A}_{k,22} \end{bmatrix}, \qquad (11)$$

are block upper triangular, where $\widetilde{A}_{k,11} \in \mathrm{R}^{\underline{n}\times\underline{n}}$, $\widetilde{A}_{k,22} \in \mathrm{R}^{(n_{k+1}-\underline{n})\times(n_k-\underline{n})}$ for $k = 1,\ldots,N$. Moreover, $\widetilde{A}_{N,11}$ is in Hessenberg form, $\widetilde{A}_{k,11}$ for $k = 1,\ldots,N-1$ are upper triangular and $\widetilde{A}_{k,22}$ for $k = 1,\ldots,N$ are upper trapezoidal.

The computation of the EPHF is bounded by $\frac{10}{3}n^3N$ *flops* and the application of the transformation

$$\widetilde{B}_k = Q_{k+1}^T B_k, \quad \widetilde{C}_k = C_k Q_k$$

involves additionally at most $2n^2(m+p)N$ *flops*. Thus, for constant dimensions the operation count for the initial preprocessing gives

$$N_{op,0} = \left(\frac{10}{3}n + 2m + 2p\right) n^2 N$$

Let $(\widetilde{F} - z\widetilde{L}, \widetilde{G}, \widetilde{H}, J)$ be the lifted descriptor system corresponding to the transformed periodic system and define $\widetilde{R} := \lambda_i\widetilde{L} - \widetilde{F}$. This matrix with reduced lower bandwidth, has the typical BABD shape shown in Fig. 1, where $N = 4$ and $n_i = 4$, $i = 1,\ldots,4$.

If we exploit the reduced bandwidth of $\widetilde{R}$, then the computation of the LU-decomposition of $\widetilde{R}$ using Algorithm 1 requires about $5n^3N$ *flops* and the forward substitution to solve $LY = P\widetilde{G}$ requires $\max\{n^2m(N-1)(N+2), n^2m\}$ *flops*. The total number of operations for $N > 1$ is

$$N_{op,f} = \left(5n^3 - 2n^2m\right)N + \left(4n^2m + 2pmn\right)N^2 \quad (12)$$

The additional storage required by this method is approximately the same as for the general BABD approach.

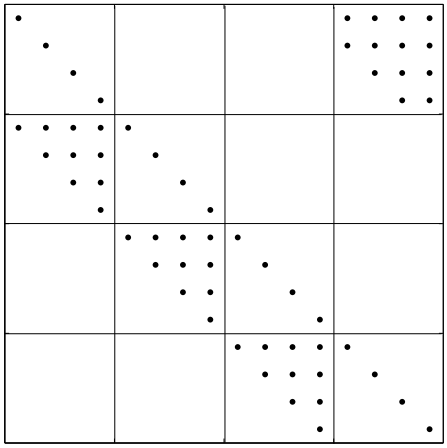

Fig. 1. BABD structured matrix $\widetilde{R}$.

B. Hessenberg method with perfect shuffle

On the matrix $\widetilde{R}$ we can perform a special permutation of the rows and columns known as *perfect shuffle*. The corresponding permutation matrix P is applied to $\widetilde{R}$, $\widetilde{G}$ and $\widetilde{H}$, to get the permuted matrices

$$\widehat{R} = P^T\widetilde{R}P, \quad \widehat{G} = P^T\widetilde{G}, \quad \widehat{H} = \widetilde{H}P,$$

Consider $\widetilde{R}$ partitioned as $\widetilde{R} = [\,\widetilde{R}_1^T,\ \widetilde{R}_2^T,\ \ldots,\ \widetilde{R}_N^T\,]^T$, where $\widetilde{R}_i \in \mathrm{R}^{n_i\times\nu}$. The *perfect shuffle* permutation matrix P corresponds to the following reordering of the rows of $\widetilde{R}$: take the first rows from each block $\widetilde{R}_i$ and move them in front of other rows; take the second rows from each block and move them behind the previous set of chosen rows; continue in this way with the third, forth, etc. rows of each block, until finished. In the case when some blocks have no more rows to be moved, then we simply skip to the next block.

The result $\widehat{R}$ of the perfect shuffle of $\widetilde{R}$ is an upper Hessenberg matrix with a very special sparse structure [10] as can be seen in Fig. 2. Ignoring the structure of the upper triangular part, the computational complexity is unacceptably large: $O(n^2N^2)$ for the LU decomposition of the Hessenberg matrix $\widehat{R}$ and $O(n^2mN^3)$ for the forward and back substitution based computation of $X = U^{-1}L^{-1}P\widehat{G}$. In this moment it is not clear how to further exploit the detailed structure of $\widehat{R}$ to reduce these figures.

The only approach which appears to be a reasonable alternative is to use sparse matrix techniques. Direct solvers for sparse matrices involve much more complicated algorithms than for dense matrices. Generally one computes the LU factorization in the form $LU = P\widehat{R}Q$, where P and Q are row and column permutation matrices, respectively. The solution is determined by forward substitution to compute $Y = L^{-1}(P\widehat{G})$ and back substitution to compute $X = QU^{-1}Y$. The main complication is due to the need to choose P and Q for efficiently handling the fill-in in

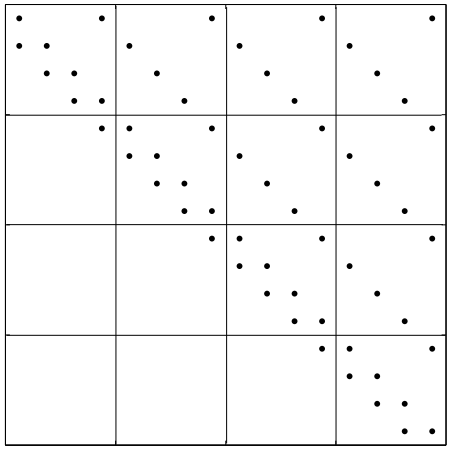

Fig. 2. Perfect shuffle $\widehat{R}$ of the BABD structured matrix $\widetilde{R}$.

the factors L and U. A typical sparse solver consists of four distinct steps (opposed to two in the dense case):

1) An ordering step that reorders the rows and columns such that the factors suffer little fill, or that the matrix has special structure, such as block-triangular form.
2) An analysis step or symbolic factorization that determines the nonzero structures of the factors and creates suitable data structures for the factors.
3) Numerical factorization that computes the L and U factors.
4) A solve step that performs forward and back substitution using the factors.

There is a vast variety of algorithms associated with each step (see for example the review paper by Duff [11]). Usually steps 1 and 2 involve only the graphs of the matrices (i.e., the zero/non-zero structure), and hence only integer operations. Steps 3 and 4 involve floating-point operations. Step 3 is usually the most time-consuming part, whereas step 4 is about an order of magnitude faster (for one vector in the right-hand side). The algorithm used in step 1 is quite independent of that used in step 3. But the algorithm in step 2 is often closely related to that of step 3 and the solver may combine steps 2 and 3 (e.g., SuperLU) or even combine steps 1, 2 and 3 (e.g., UMFPACK) so that the numerical values also play a role in determining the elimination order [11].

It is well known that a complete *a priori* analysis of the computational complexity is impossible since the work in a sparse LU factorization depends on numerical pivoting choices and the efficacy of additional heuristic column reordering intended to minimize fill-in. Efficient algorithms to compute the sparse LU decomposition and perform forward and backward substitutions have the property that their computational complexity is $O(\tau) + O(\mu)$, where $\tau = \frac{n(n+3)}{2}N$ is the number of nonzero elements in $\widehat{R}$ and $\mu = nmN^2$ is the order of the problem [11]. Moreover, sparse representation of data allows to perform the matrix multiplications involved to evaluate $HX+J$ exploiting the sparse block-diagonal structures of H and J. The required additional storage covers the index information associated to store sparse matrices and the unavoidable *fill-in*.

C. Schur method with perfect shuffle

Instead of reduction to the EPHF, it is possible to reduce the periodic matrix A_k to the *extended periodic real Schur form* (EPRSF) using an N-periodic orthogonal transformation [9]. In the EPRSF, the reduced matrices $\widetilde{A}_k$ have the form (11), with the only difference that $A_{N,11}$ is in quasi-upper triangular form having only 1×1 or 2×2 diagonal blocks. The 2×2 blocks correspond to complex conjugated eigenvalues of the product $A_N \cdots A_2 A_1$. For constant dimensions, the cost of the preprocessing is

$$N_{op,0} = (25n + 2m + 2p)\, n^2 N$$

which is significantly larger than the cost to compute the EPHF. Note however, that this operation is performed only once.

The result $\widehat{R}$ of the perfect shuffle of $\widetilde{R}$ is a block upper triangular matrix, with the diagonal blocks in unreduced Hessenberg form [10]. For constant dimensions, the block sizes of the Hessenberg matrices are either N or $2N$. A block of order $2N$ results for each 2×2 diagonal block of $A_{N,11}$. For large periods (e.g., $N \geq 500$), the diagonal submatrices are still large and sparse. The equation $\widehat{R}X = \widehat{G}$ can be solved by a block back substitution, factoring only the diagonal blocks of the matrix $\widehat{R}$. The following is a general block back substitution algorithm to solve $UX = Y$, where we assume U is block upper triangular with r diagonal blocks and X and G are accordingly partitioned row-wise in r block-rows of dimension $N \times M$ or $2N \times M$:

Algorithm 4: *Block-back substitution to solve* $UX = Y$.

for $i = r, \ldots, 1$

Solve $U_{ii}X_i = Y_i - \sum_{j=i+1}^{r} U_{i,j}X_j$

end

When employing this algorithm to solve $\widehat{R}X = \widehat{G}$, the fine-grain structure of both $\widehat{R}$ and $\widehat{G}$ can be further exploited. For example, assuming all blocks are $N \times N$ (thus $r = n$), forming $U_{i,j}X_j$ involves at most $2mN^2$ *flops*. Thus, the number of operations to form the right-hand sides in **Algorithm 4** is at most mn^2N^2, while solving the equations with sparse upper Hessenberg U_{ii} involves additionally $O(nN)$+$O(mn^2N^2)$ *flops*.

D. Fast method

To evaluate $\widehat{W}(\lambda_i)$, we can apply the method of Laub [12] to the standard lifted representation (5). The preliminary computation involves building the system matrices followed by the reduction of $\widehat{F}$ to the Hessenberg form and application of transformations to $\widehat{G}$ and $\widehat{H}$. To build the system matrices, the following algorithm can be used:

Algorithm 4: *Building* $(\widehat{F},\widehat{G},\widehat{H},\widehat{J})$.

Comment. Build $\widehat{F}$ and $\widehat{H}$.

$\widehat{F}=A_1$, $\widehat{H}=C_1$.

for $i=2,\ldots,N$

$$\widehat{H}\leftarrow\left[\begin{array}{c}\widehat{H}\\ C_i\widehat{F}\end{array}\right],\ \widehat{F}\leftarrow A_i\widehat{F}$$

end

Comment. Build $\widehat{G}$ and $\widehat{J}$.

$\widehat{G}=B_1$, $\widehat{J}_{11}=D_1$.

for $i=2,\ldots,N$

$[\,\widehat{J}_{i,1}\ \cdots\ \widehat{J}_{i,i-1}\,|\,\widehat{J}_{i,i}\,]=[\,C_i\widehat{G}\,|\,D_i\,]$

$\widehat{G}\leftarrow[\,A_i\widehat{G}\ B_i\,]$

end

For constant dimensions, **Algorithm 4** requires about $(n^2m+nmp)N^2+(2n^3+2pn^2-mn^2-mnp)N$ real *flops*. Additionally, the reduction to the Hessenberg form $\widetilde{F}=Q^T\widehat{F}Q$ requires $\frac{10}{3}n^3$ *flops* and the application of transformations $\widetilde{G}=Q^T\widehat{G}$ and $\widetilde{H}=H\widehat{Q}$ requires additionally $2Nn^2(m+p)$. It follows that the total number of preliminary operations is

$$N_{op,0}=(n^2m+nmp)N^2+(2n^3+(4p+m)n^2-npm)N+\frac{10}{3}n^3$$

The evaluation of $\widehat{W}(\lambda_i)=\widetilde{H}(\lambda_i I_{n_1}-\widetilde{F})^{-1}\widetilde{G}+\widehat{J}$ requires

$$N_{op,f}=2N^2pmn+Nmn^2$$

The additionally needed storage to build the matrices of the lifted system is

$$N_{st}=n^2+(nm+pn)N+pmN(N+1)/2$$

We can substantially save operations if we reduce first the periodic state matrix A_k to an EPHF. In this case, all multiplications to build the lifted system matrices $(\widehat{F},\widehat{G},\widehat{H},\widehat{J})$ which involve upper triangular or Hessenberg matrices, require practically the half of operations of those for full matrices. The total number of preliminary operations is

$$N_{op,0}=\left(\frac{n^2m}{2}+nmp\right)N^2+\left(\frac{11}{3}n^3+3(p+\frac{m}{2})n^2-npm\right)N$$

while for the evaluation of $\widehat{W}(\lambda_i)$ the same number of *flops* is necessary.

E. Transfer-function based computation

We can evaluate first the TFM $W(z)$ using the method proposed in [13] and then compute $W(\lambda_i)$ for $i=1,\ldots,N_f$. The main part of the computational effort is the evaluation of $W(z)$, which for constant dimensions is of order $N_{op,0}=O(N^3pmn^3)$. The effort to compute $W(\lambda_i)$ with $W(z)$ given in a TFM form is $N_{op,1}=O(N^2pm)$. Thus, this method can be an alternative to other methods only for very large N_f.

IV. Numerical experiments

We performed several numerical tests for standard periodic systems with constant dimensions using prototype implementations in MATLAB of the proposed approaches. The obtained timing results on a Pentium IV 3 GHz machine reflect only very approximately the real complexity of the computations, being obtained with implementations which are far to be optimal with respect to speed and/or memory usage. The following algorithms have been tested:

FULL	GEPP without structure exploitation
GEPP	sparse GEPP with BABD structure
GEPP-H	sparse GEPP with periodic Hessenberg form
GEPP-S	sparse GEPP with periodic Schur form
FAST	fast method based on standard lifting

The implementations of GEPP, GEPP-H and GEPP-S are based on the sparse matrix linear equation solvers available in MATLAB. In all computations, the resulting frequency responses have a relative accuracy of order 10^{-9} at least.

The results have been obtained for randomly generated periodic systems with $m=3$ inputs and $p=6$ outputs. A first set of systems with large periods N but relatively small state dimension $n=6$ has been considered. In Table I we present CPU timing results (in seconds) obtained for different periods for a single evaluation of $W(\lambda_i)$:

N	50	100	200	500
FULL	0.12	0.64	13.9	88.3
GEPP	0.05	0.20	0.96	5.13
GEPP-H	0.07	0.24	0.86	5.13
GEPP-S	0.12	0.43	1.74	12.64
FAST	0.016	0.031	0.094	0.45

TABLE I

TIMING RESULTS FOR LARGE PERIODS (IN SECONDS)

Note that for $N=500$, the frequency response $W(\lambda_i)$ is a 3000×1500 complex matrix. As can be seen, structure exploiting allows to compute this matrix in a reasonable time on a desktop computer. Interestingly for this category of problems, the additional structure added by employing the periodic Hessenberg or Schur forms brings practically no benefits in speed over the non-reduced case. The time requirements to compute either the periodic Hessenberg or Schur forms are practically negligible (e.g., 0.125 and 0.141 seconds respectively for $N=500$). Similarly, the effect of employing column interchanges to reduce *fill in* was not significant, and occasionally even increased the execution times.

We computed also with a second category of examples with a small period of $N=10$, but relatively large state dimensions. In Table II we present CPU timing results (in seconds) obtained for different state dimensions. As can be observed, the reduction to Hessenberg or Schur form have the clear consequence of reducing the computational times, although the use of Hessenberg form led to smaller

execution times than using the Schur form. However since these implementations are just prototype codes based on general purpose sparse linear system solvers (thus without explicitly exploiting the simplified BABD structure of the linear systems to be solved), this timing results should not be overemphasized.

n	30	60	120	300
FULL	0.047	0.27	1.7	23.64
GEPP	0.062	0.50	6.36	374.28
GEPP-H	0.047	0.20	2.0	63.0
GEPP-S	0.057	0.32	2.77	75.82
FAST	0.003	0.006	0.019	0.18

TABLE II

TIMING RESULTS FOR LARGE STATE DIMENSIONS (IN SECONDS)

V. DISCUSSION

We presented several methods to evaluate the frequency-response of linear periodic discrete-time systems. Structure exploitation allows to compute more efficiently the frequency-response for systems of large orders and/or large periods. For standard systems, the method of reference is Laub's method [12] which has a computational complexity of $O(n^3)+O(mn^2)+O(pn^2)$ for the preprocessing phase and $O(mn^2) + O(pmn)$ for the evaluation for a single frequency value. In the case of periodic systems, the TFM of the lifted system is a $(pN) \times (mN)$ matrix. It follows that the ideal computational complexity for the preprocessing phase should be $O(n^3N)+O(mn^2N^2)+O(pn^2N^2)$. This order of magnitude is practically achieved by all presented methods, if we assume $\max(m,p) \leq n$ for the "fast" method where the term $nmpN^2$ is also present. The extrapolated ideal computational complexity for the evaluation phase should be $O(mn^2N)+O(pmnN^2)$. This complexity is achieved only by the "fast" method, the rest of methods having complexity terms of the form $O(n^3N)$ and $O(n^2mN^2)$. This difference in the computational complexity is indirectly confirmed by the timing results presented in the previous section.

In light of these considerations, for standard periodic systems, the "fast" method based on using the standard lifted representation in conjunction with the standard Hessenberg approach appears to be a satisfactory algorithm from the point of view of computational complexity. Note however, that the generation of the lifted model requires forming explicitly products of matrices, which, besides the undesirable accumulation of roundoff errors, can lead for large periods, to severe numerical difficulties (e.g., overflows). An important problem to be additionally considered is how to monitor the conditioning of the linear systems to be solved (to avoid ill-conditioned systems for certain values of λ).

The Hessenberg, Schur and "fast" methods for standard periodic systems can be extended to compute the frequency-response of descriptor periodic systems, by employing appropriate preliminary similarity transformations. More details on these approaches will be provided in a companion paper (in preparation).

VI. ACKNOWLEDGMENTS

The work of the author has been performed in the framework of the Swedish Strategic Research Foundation Grant "Matrix Pencil Computations in Computer-Aided Control System Design: Theory, Algorithms and Software Tools".

REFERENCES

[1] O. M. Grasselli and S. Longhi, "Finite zero structure of linear periodic discrete-time systems," *Int. J. Systems Sci.*, vol. 22, pp. 1785–1806, 1991.

[2] R. A. Meyer and C. S. Burrus, "A unified analysis of multirate and periodically time-varying digital filters," *IEEE Trans. Circuits Syst.*, vol. 22, pp. 162–168, 1975.

[3] O. M. Grasselli and S. Longhi, "Zeros and poles of linear periodic discrete-time systems," *Circuits, Systems and Signal Processing*, vol. 7, pp. 361–380, 1988.

[4] P. Amodio, J. R. Cash, G. Roussos, R. W. Wright, G. Fairweather, I. Gladwell, G. L. Kraut, and M. Paprzycki, "Almost block diagonal linear systems: sequential and parallel solution techniques, and applications," *Numerical Linear Algebra with Applications*, vol. 7, pp. 275–317, 2000.

[5] G. Fairweather and I.Gladwell, "Algorithms for almost block diagonal linear systems," *SIAM Review*, vol. 46, pp. 49–58, 2004.

[6] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Baltimore: John Hopkins University Press, 1989.

[7] S. Wright, "A collection of problems for which gaussian elimination with partial pivoting is unstable," *SIAM J. Scientific and Statistical Computing*, vol. 14, pp. 231–238, 1993.

[8] A. W. Bojanczyk, G. Golub, and P. Van Dooren, "The periodic Schur decomposition. Algorithms and applications," in *Proceedings SPIE Conference*, F. T. Luk, Ed., vol. 1770, July 1992, pp. 31–42.

[9] A. Varga, "Balancing related methods for minimal realization of periodic systems," *Systems & Control Lett.*, vol. 36, pp. 339–349, 1999.

[10] D. Kressner, "Numerical methods and software for general and structured eigenvalue problems," Ph.D. dissertation, TU Berlin, Institut für Mathematik, Berlin, Germany, 2004.

[11] I. Duff, "Direct methods," Technical report RAL-TR-1998-054, Rutherford Appleton Laboratory, Chilton, Didcot, OX11 0QX UK., 1998. [Online]. Available: citeseer.ist.psu.edu/duff98direct.html

[12] A. J. Laub, "Efficient multivariable frequency response computations," *IEEE Trans. Automat. Control*, vol. 26, pp. 407–408, 1981.

[13] A. Varga, "Computation of transfer functions matrices of periodic systems," *Int. J. Control*, vol. 76, pp. 1712–1723, 2003.