FEATURE

Solving Large-Scale Control Problems

© EYEWIRE

Sparsity and parallel algorithms: two approaches to beat the curse of dimensionality.

By Peter Benner

In this article we discuss sparse matrix algorithms and parallel algorithms, as well as their application to large-scale systems. For illustration, we solve the linear-quadratic regulator (LQR) problem and apply balanced truncation model reduction using either parallel computing or sparse matrix algorithms. We conclude that modern tools from numerical linear algebra, along with careful investigation and exploitation of the problem structure, can be used to derive algorithms capable of solving large control problems. Since these approaches are implemented in production-quality software, control engineers can employ complex models and use computational tools to analyze and design feedback control laws.

Background

Classic and modern control theory is usually concerned with analyzing and synthesizing systems described by ordinary differential equations (ODEs) that often represent physical laws of motion. The linearization of higher order ODEs, such as second-order systems describing classical mechanical systems, or linearization of nonlinear ODEs about an equilibrium, leads to the well-known representation of a linear-time invariant (LTI) system

0272-1708/04/$20.00©2004IEEE

$$\dot{x}(t) = Ax(t) + Bu(t), \quad t > 0, x(0) = x_0,$$
$$y(t) = Cx(t) + Du(t), \quad t \geq 0, \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, and $D \in \mathbb{R}^{p\times m}$ are constant matrices. The order n of the system ranges from a few tens to several hundred as in control problems for large flexible space structures. The increase in dimension and the desire to control multi-input/multi-output systems triggered the need for reliable standard software from numerical linear algebra packages. These packages include LINPACK/EISPACK in the 1970s or LAPACK [1] in the late 1980s and 1990s. Several approaches to building software libraries for control based on those packages have been pursued; see [2] for a review. The most successful package identified for control-oriented computations is MATLAB, together with its different toolboxes related to control [3]. The basic computational kernels in MATLAB were also based on LINPACK/EISPACK and from version 6.0 on they are based on LAPACK. With the availability of LAPACK-based implementations in MATLAB and SLICOT, the subroutine library in control theory [2], it is now possible to apply analysis and synthesis methods to systems of the form (1) with order up to $n = 1,000$.

At the same time, the complexity of models describing physical problems has also increased, especially systems modeled by partial differential equations (PDEs). The design and analysis of controllers and observers is challenging for these systems, and there has been considerable effort devoted to optimal control [4]. Here we focus on the LQR problem for parabolic PDEs [5]. A comprehensive treatment using the general framework of linear evolution equations and semigroup theory yields systems of the form (1) with matrices replaced by linear (infinite-dimensional) operators; see [6], [7]. Many practically relevant PDEs are analyzed in detail in [7], for example, Euler–Bernoulli, Kelvin–Voigt, or Kirchhoff equations. Although an approximation theory has been developed, the issue of numerical methods used to *solve* these problems has been inadequately treated. The lack of adequate numerical methods and software frequently prevents the use of modern control methodologies such as linear quadratic Gaussian (LQG) or H_∞ controller design when the underlying physical problem is modeled by PDEs. Instead of using control and system-theoretic concepts, the numerical analysis approach to solving PDE-constrained optimal control problems is based on the fully discretized PDE and uses optimization algorithms such as Newton's method and quadratic programming to solve the subsequent optimization problem. In the next section we discuss some of the obstacles to applying standard control methodologies to PDE control problems.

Large-Scale Control Problems

A major source of large-scale control problems is the modeling of the underlying physical process by PDEs. Since the LQR framework is analogous to the finite-dimensional case [6], [7], this approach is attractive for infinite-dimensional systems.

A major source of large-scale control problems is the modeling of the underlying physical process by partial differential equations.

What are the computational problems that arise in controlling PDEs? Discretizing the partial derivatives for spatial coordinates (and possibly linearizing the resulting ODE for the mesh function) leads again to a system of the form (1). The system dimension easily exceeds 10,000 for problems in two space dimensions (2-D) and may reach several million for three-dimensional (3-D) problems, although the number of inputs and outputs remains small, that is, $m, p \ll n$. Therefore, methods used for standard state-space systems arising from ODE models, which are mostly of computational complexity $\mathcal{O}(n^3)$ floating point operations (flops) and require $\mathcal{O}(n^2)$ workspace, can no longer be applied for such models. For instance, using a standard 2-D array for storing the coefficient matrix of the Laplace operator (one of the simplest partial differential operators, arising, for example, in heat conduction problems), defined on a 3-D cube and discretized using 100 grid points in each direction, requires 745 Gb if the IEEE double precision standard is used for floating point numbers and still half as much for single precision. Even if we could store the matrix in the main memory, solving a linear system of equations for this coefficient matrix with Gaussian elimination, which requires roughly $2n^3/3$ flops, would take 105 years on an Intel Pentium 4 processor running at 2 GHz. Since the discretized Laplace operator is a sparse matrix, most matrix entries are zero and need not be stored (more than 99.9% of the matrix entries are zero!).

Fortunately, there exist efficient algorithms for exploiting this special sparsity structure. Using modern sparse linear algebra techniques, the problem discussed above can be solved in seconds on the Pentium 4 machine. Hence, *solving* PDEs is a problem for which advanced techniques exist, and many challenging numerical problems have been solved in the last decades. However, *controlling*

or *regulating* a system governed by a PDE is more difficult. The usual analysis and synthesis tools available, for example, in the MATLAB Control Systems Toolbox or even the algorithmically more advanced SLICOT library, have limited ability to exploit sparse structures. Using recently developed tools for solving the underlying matrix equations for sparse coefficient matrices, we claim that PDE control is possible using modern controller synthesis. This claim is demonstrated below for the LQR problem applied to the heat equation.

Another approach for treating large-scale problems is to reduce the size of the problem using model-reduction techniques. Large problems with dense coefficient matrices can also arise, such as when the boundary element method or wavelet techniques are used to semidiscretize the PDE. The only way to solve the resulting control problems is to increase the computing power as algorithms of complexity n^3 are needed. Reducing the complexity of these algorithms by employing the multiscale or hierarchical structure of the resulting system matrix is a topic of ongoing research; see [8]. Here, we discuss the use of parallel computing, which exploits the computing power and memory of multiple computers. Recent advances in cluster computing, network technology, and parallel algorithms allow large-scale problems to be solved using only off-the-shelf hardware components (such as PCs available in a lab connected in a local area network (LAN)), and standard software components for communication as well as for computation. We shall demonstrate how traditional algorithms from system analysis and synthesis such as solving linear (Lyapunov or Sylvester) matrix equations, stabilizing a system, and computing LQR controllers, can be applied to systems with thousands of states if the right algorithms from parallel linear algebra are employed. Such parallel computation techniques can also serve as black-box tools to test methodologies for large-scale sparse systems before implementing sparse linear algebra techniques, a tedious and complicated task that should be done only after determining which synthesis methods are to be applied (for example, discrete-time or continuous-time model description and H_2- or H_∞-control).

Sparsity and Parallelization

Sparse and parallel matrix algorithms are not competitive, since they aim at different problem classes. For large-scale sparse problems, parallel computing eventually becomes necessary. Therefore, a future goal is to combine both techniques to make the methods applicable to control problems for PDEs on complex 3-D computational domains.

Sparse Matrices

A matrix $A \in \mathbb{R}^{n\times n}$ is sparse if most of its entries are zero. If the number of nonzero entries is denoted by nz, then we say that a matrix is sparse whenever $nz = \mathcal{O}(n)$.

Origin of Sparse Matrices

Sparse matrices typically arise in the discretization of partial differential equations. Consider the Poisson equation $-\Delta\mathbf{x}(\xi,\eta) = \mathbf{f}(\xi,\eta)$ for $(\xi,\eta) \in \Omega = [0,1]\times[0,1]$, with homogeneous Dirichlet boundary conditions $\mathbf{x} \equiv 0$ on $\partial\Omega$. Here, $\Delta := (\partial^2/\partial\xi^2) + (\partial^2/\partial\eta^2)$ is the Laplace operator. Using a standard finite-differences method (FDM) with uniform grid of mesh size $h = 1/(N+1)$ in both space directions yields a linear system of equations $Ax = f$ in R^{N^2}. Here, $f_{N(i-1)+j} = \mathbf{f}(ih, jh)$, $x \in \mathbb{R}^{N^2}$ is the vector of unknowns, where $x_{N(i-1)+j}$ is the computed approximation to $\mathbf{x}(ih, jh)$, and

$$A = \begin{bmatrix} T_N & -I_n & & \\ -I_n & \ddots & \ddots & \\ & \ddots & \ddots & -I_n \\ & & -I_n & T_N \end{bmatrix},$$

$$T_N = \begin{bmatrix} 4 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{bmatrix}. \tag{2}$$

Here, I_n or just I denotes the identity matrix in $\mathbb{R}^{n\times n}$.

As a consequence of the FDM approach, each row and each column of A has at most five nonzero entries. That is, only $(5/N^2)\times 100\%$ of the matrix entries need to be stored. For $N = 100$, the required storage space (using IEEE double precision format) of 763 MB for dense matrices is reduced to less than 600 kB for the sparse matrices in (2). The exact storage requirements depend on the storage format used; see [9] for a discussion of several sparse matrix formats. For (2), further reduction is achieved by observing that the nonzero entries are constant along five diagonals (except for some intervening zeros). However, this structure does not hold even for slightly more complex differential operators, domains, or discretizations.

The above observations relate to (1) as follows. Suppose the control problem consists of regulating the temperature in the square domain Ω using a heat source in the midpoint of the square. Then the governing equation is the (instationary) heat-diffusion equation with point control

$$\frac{\partial}{\partial t}\mathbf{x} = \alpha\Delta\mathbf{x} + \mathbf{B}u \quad \text{in } \Omega$$

with an appropriately chosen operator $\mathbf{B}$, a constant $\alpha > 0$, and $u \in L_2[0,\infty)$. With homogeneous Dirichlet boundary conditions and for a given initial temperature distribution $g(\xi,\eta)$, the same FDM discretization applied to Δ and N odd yields the stable single-input LTI system

$$\dot{x}(t) = -\alpha Ax(t) + Bu(t), \tag{3}$$

where $x_{N(i-1)+j}(t)$ is the approximation to $\mathbf{x}(t, ih, jh)$, A is given by (2), and $B = e_{((N+1)^2/2)}$, where e_k denotes the kth unit vector in $\mathbb{R}^n$. With a suitably chosen output equation $y = Cx$, we thus obtain a standard linear system as in (1).

Many methods for system analysis and control design involve a state-space transformation. These transformations destroy the sparsity structure and produce a dense matrix that requires workspace for $n^2 = N^4$ real numbers. For the example mentioned above ($N = 100$), this situation represents a severe problem even on high-end workstations. Also, the computational complexity of $n^3 = N^6$ flops of most analysis and synthesis methods prevents the application of modern control-theoretic methods. On the other hand, solving the Poisson equation with a sparse method can be achieved in seconds on desktop computers. Therefore, modern control techniques can only be applied if we are able to reduce the complexity of the necessary algorithms to $\mathcal{O}(n \log^k n)$ flops for some modest integer k so that they nearly scale with the problem size n.

Another approach for treating large-scale problems is to reduce the size of the problem using model-reduction techniques.

Sparse Matrix Algorithms

There are two different directions for exploiting sparse structures: sparse direct methods and iterative methods. Iterative methods such as Krylov subspace methods (see [9]–[11]) have the advantage that they do not alter A during the computations; the sole information about A required and the major computational task in each iteration is the result of a matrix-vector product. Exploiting sparsity, the matrix-vector product Az for $z \in \mathbb{R}^n$ can be implemented in $\mathcal{O}(nz) = \mathcal{O}(n)$ flops. If the iterative process converges in $k \ll n$ steps to the solution of the linear system of equations, then we have achieved the desired complexity reduction. On the other hand, direct methods (see [12] for an introduction to the topic) are specially adapted versions of Gaussian elimination or LU/Cholesky factorization that minimize the level of fill-in during the elimination process. By fill-in we mean the generation of new nonzero entries in the matrix A during the elimination process. These methods carefully analyze the sparse matrix and try to find a permutation such that the fill-in during the elimination process for the permuted matrix is minimized. Many other details are needed for a successful application of sparse direct methods; see [12]. If sparse direct solvers are applicable in an efficient way so that linear systems can be solved in $\mathcal{O}(\gamma \cdot nz)$ flops with a modest constant γ (as is the case in problems arising from discretized partial differential operators), then these methods are usually preferred over iterative solvers due to numerical stability and the ability to avoid stopping criteria.

The key to solving large-sparse control problems obtained from semidiscretization of PDEs using FDM or the finite element method (FEM) is to replace the $\mathcal{O}(n^3)$ algorithms by methods that rely on sequences of solutions of linear systems, where we can use direct or iterative methods. These methods are well developed in the sparse numerical linear algebra community, and reliable software is available. Applications to Lyapunov and algebraic Riccati equations enable LQR design and model reduction techniques for large systems. These techniques carry over to H_2 and H_∞ optimization.

There are efforts aimed at applying Krylov subspace methods to large-scale control problems using projection techniques. This technique can be described as follows. The control state space is projected onto a low-dimensional subspace given by an appropriate basis for the computed Krylov subspace. This projection in itself can be seen as a model reduction technique (see also the section "Model Reduction"). The control problem is solved in the low-dimensional setting, and the solution is prolongated to full scale. Despite recent efforts in this direction, this approach cannot be considered mature as far as control system design is concerned. For example, if this technique is employed for the Riccati approach to solving LQR/LQG design problems, there is no way to guarantee closed-loop stability for the full-scale problem (see also the discussion in "The LQR Problem"). It should be noted, though, that Krylov subspace techniques are successful as model reduction techniques, in particular in circuit simulation; see the recent reviews [10] and [13]. There remain important open questions related to control system design, to which we shall return later.

Parallel Matrix Algorithms

A relatively new approach for dealing with large-scale problems is to use the resources of many processors. Clusters constructed from commodity systems (PCs and LANs with local area switches) and "open" software have started to be widely used for parallel scientific computing. Notable improvements in the performance of commodity hardware have resulted in affordable tools for large-scale scientific applications. Cluster computing has become a major research field in scientific computing, as emphasized by the annual IEEE Cluster Computing Conference, conferences and workshops dedicated to this topic, and

special issues of leading scientific journals on parallel computing. For really large problems, we have to resort to distributed memory computing in contrast to a shared memory environment, which permits the use of an arbitrary amount of workspace consisting of the main memory of all computers in the network forming the cluster.

The major ingredient in the success of a parallel algorithm is the availability of affordable hardware, portable software for data communication such as MPI (message-passing interface) [14], and portable software for basic linear algebra computations, such as those available in the BLAS and LAPACK or ScaLAPACK [15]. These libraries enhance reliability and improve portability of the routines since we use the most general parallelization strategy not targeted to a specific computing platform. ScaLAPACK is based on the PBLAS (a parallel version of the serial BLAS) for basic computations such as scalar and matrix products and employs communication subroutines collected in a sublibrary called BLACS (Basic Linear Algebra Communication Subroutines). The BLACS provide routines for passing matrices and vectors between processors and can be ported to any serial and parallel architecture with an implementation of MPI.

In ScaLAPACK, the computations are performed on a logical rectangular grid of $n_p = p_r \times p_c$ processes. The processes are mapped onto the physical processors, depending on their availability. All data matrices have to be distributed among the process grid prior to the invocation of a ScaLAPACK routine. It is the user's responsibility to perform this data distribution. Specifically, in ScaLAPACK the matrices are partitioned into $mb \times nb$ blocks, and these blocks are then distributed and stored among the processes (see [15] for details).

Sparse matrix algorithms often have a high potential for parallelization. This topic is of high priority for future research. Here, we only discuss how parallel computing enables us to employ techniques that have been successfully applied in control system design for large control problems.

In view of parallelizing the computational algorithms underlying analysis and synthesis routines for control systems, we face a major difficulty. The workhorse of *all* functions for controller design based on the LQR/LQG or H_∞ formalism in the MATLAB Control Toolbox and also in SLICOT is the QR algorithm, which yields the (real) Schur form of a matrix $A \in \mathbb{R}^{n\times n}$ by computing an orthogonal matrix $U \in \mathbb{R}^{n\times n}$ and a quasi-upper triangular matrix T (the Schur form *of* A) such that $AU = UT$. By arranging the eigenvalues of A in a desired order on the diagonal of T, we obtain a corresponding A-invariant subspace (an A-invariant subspace $\mathcal{S} \subset \mathbb{R}^n$ is defined by the property $A\mathcal{S} \subset \mathcal{S}$) spanned by the leading k columns of U corresponding to the desired eigenvalues. The QR algorithm is the major ingredient in the Bartels–Stewart method or Hammarling's method for solving Lyapunov equations as well as in the Schur vector method for solving algebraic Riccati equations; see [16]. These fundamental matrix equations have to be solved for LQR/LQG/H_2 and H_∞ controller design as well as in model reduction methods based on balancing. Unfortunately, the QR algorithm has proved to be a challenging problem in the development of parallel matrix algorithms [17]. A parallel implementation of the QR algorithm is available in ScaLAPACK, but its performance remains unsatisfactory.

Recent attempts in parallel algorithms for control problems have focused on replacing the QR algorithm with another general purpose algorithm that can serve as the basis for solving Lyapunov and Riccati equations. In fact, the Schur form provides more information than is needed in the control problems mentioned above. Specifically, while the Schur form separates each eigenvalue from the others, it suffices to separate the stable eigenvalues (those in the open left-half plane) from the unstable ones and find a corresponding invariant (or deflating) subspace. The A-invariant subspace $\mathcal{V}$ corresponding to the stable eigenvalues is called the stable invariant subspace and can be computed from a projector P onto $\mathcal{V}$. Here, P is a projector onto $\mathcal{V}$ if and only if range $(P) = \mathcal{V}$ and $P^2 = P$. An orthogonal basis of $\mathcal{V}$ can then be computed by means of a QR factorization of P. Spectral projection methods compute such a projector P. Since the problem of solving Lyapunov and Riccati equations is closely related to invariant subspace computation, these methods can be applied in this context. The most popular of these methods is the sign function method [18], which we briefly describe in the following.

The Sign Function Method

Consider a matrix $Z \in \mathbb{R}^{n\times n}$ with no eigenvalues on the imaginary axis $j\mathbb{R}$. Let

$$Z = S \begin{bmatrix} J^- & 0 \\ 0 & J^+ \end{bmatrix} S^{-1}$$

be its Jordan decomposition, where $J^- \in \mathbb{R}^{k\times k}$ and $J^+ \in \mathbb{R}^{(n-k)\times(n-k)}$ contain the Jordan blocks corresponding to the stable and antistable, respectively, parts of the spectrum of Z. The matrix sign function of Z is defined as

$$\operatorname{sign}(Z) := S \begin{bmatrix} -I_k & 0 \\ 0 & I_{n-k} \end{bmatrix} S^{-1}.$$

Note that sign(Z) is uniquely defined [19]. Alternative definitions of the sign function can be given; see [20] for an overview. Computing the sign function of a matrix yields a spectral projection method since $(I_n - \operatorname{sign}(A))/2$ defines a projector onto the stable A-invariant subspace. In the sequel this property is used to solve several computational problems.

Computing sign(Z) can be achieved as follows: applying Newton's method to the equation $Z^2 - I_n = 0$, we obtain the classical iteration for the matrix sign function:

$$Z_0 \leftarrow Z, \quad Z_{j+1} \leftarrow \frac{1}{2}\left(c_j Z_j + \frac{1}{c_j} Z_j^{-1}\right), \qquad j = 0, 1, 2, \ldots, \tag{4}$$

where $c_j \in \mathbb{R}$ is an acceleration parameter. Under the given assumptions, the sequence $\{Z_j\}_{j=0}^{\infty}$ converges to sign(Z) with a locally quadratic convergence rate [18]. A typical convergence history based on $\|Z_{j+1} - Z_j\|_1 / \|Z_{j+1}\|_1$ is displayed in Figure 1, showing the fast quadratic convergence rate. Here, we compute the sign function of a dense matrix A associated with the steel cooling example below with $n = 1357$. The eigenvalue of A closest to the $j\mathbb{R}$ is $\approx -6.7 \cdot 10^{-6}$. The eigenvalue of largest magnitude is ≈ -5.8, thus the condition of A is about 10^6 since A is symmetric negative definite. Usually, 8–12 steps are enough to obtain maximal accuracy. Due to the small stability margin, relatively many (17) iterations of (4) are needed. The computations were performed using MATLAB 6.5 on a Linux PC (AND Athlon processor, running at 1.7 GHz).

A major appeal of using sign function-based algorithms in parallel computing is motivated by the fact that matrix addition is an operation that requires no communication using the ScaLAPACK matrix distribution, and the computation of Z_j^{-1} using Gauss–Jordan elimination is also efficiently parallelizable; see [21] for parallel performance of the sign function method. The sign function method is criticized for several reasons, the most prominent one being the need to compute an explicit inverse at each step. The performance of the method decreases when eigenvalues close to the imaginary axis are present, but numerical instabilities basically show up only when there are eigenvalues whose imaginary parts have magnitude less than the square root of the machine precision. Hence, significant problems can be expected in double precision arithmetic (as used in MATLAB) for imaginary parts of magnitude less than 10^{-8}. A more precise analysis involves the reciprocal of the separation of stable and antistable invariant subspaces; the distance of eigenvalues to the imaginary axis is an upper bound for the separation. Fortunately, in the control applications considered here, poles are sufficiently far from the imaginary axis. On the other hand, if we have no problems with the spectral dichotomy, then the sign function method solves a spectral decomposition problem that is usually better conditioned than the Schur vector approach, as it only separates the stable from the antistable subspace while the Schur vector method essentially requires the separation of n subspaces from each other. The difference in the conditioning of the Schur form and a block triangular form (as computed by the sign function) is discussed in [22]. Therefore, counter

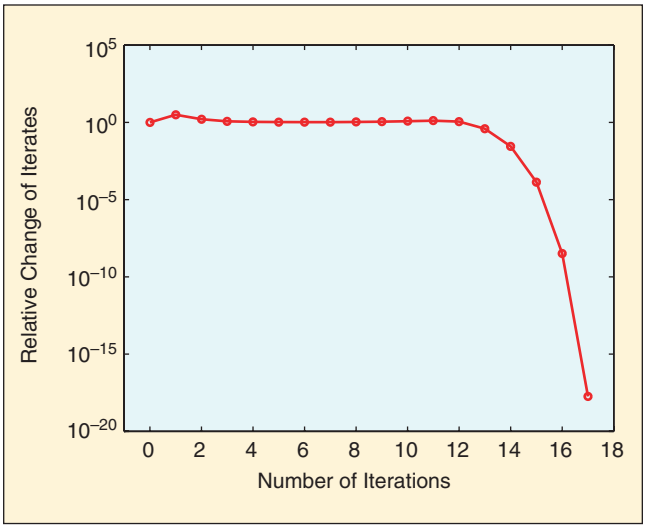

Figure 1. *Convergence history for sign(A) using (4). The initial phase of almost stagnation is relatively long in this example. The ultimate quadratic convergence rate is clearly visible.*

to intuition, it should not be surprising that results computed by the sign function method are often more accurate than when using Schur-type decompositions; examples are presented in [23].

In the following sections the sign function (iteration) is used to implement efficient parallel algorithms for solving Lyapunov and algebraic Riccati equations. As sparsity is lost in the first step of (4), the sign function method is a dense matrix technique, and algorithms based on it require $\mathcal{O}(n^3)$ flops, no matter what nz is. Recent approaches to data-sparse representations of matrices can be used to implement (4) in $\mathcal{O}(n \log^k n)$ flops. In such a setting, the sign function method becomes perfectly scalable, so that its parallel performance is retained when adding additional processors. Applications of this technique to control problems are under development.

The LQR Problem

We consider the numerical solution of the continuous-time autonomous linear-quadratic optimal control problem, known as the LQR problem:

Minimize

$$\mathcal{J}(x^0, u) = \frac{1}{2} \int_0^{\infty} (y(t)^T S y(t) + u(t)^T R u(t))\, dt \tag{5}$$

over $u \in L_2[0, \infty)$ subject to the dynamics given by (1).

For notational convenience, we assume $D = 0$. Moreover, $S = S^T \in \mathbb{R}^{p \times p}$ and $R = R^T \in \mathbb{R}^{m \times m}$ are assumed to be positive definite. Under the assumptions that (A, B) is stabilizable and (A, C) is detectable, the optimal control is given by the feedback law

$$u_*(t) := -R^{-1}B^T P_* x(t), \qquad t \geq 0, \tag{6}$$

where P_* is the stabilizing solution of the algebraic Riccati equation (ARE)

$$0 = \mathcal{R}(P) := C^T SC + A^T P + PA - PBR^{-1}B^T P, \tag{7}$$

that is, all eigenvalues of $A - BR^{-1}B^T P_*$ are in the open left-half plane.

The solution methods for the ARE can be divided into two classes. The methods in the first class are based on the connection to the eigenvalue problem for the Hamiltonian matrix

$$H = \begin{bmatrix} A & BR^{-1}B^T \\ C^T SC & -A^T \end{bmatrix}. \tag{8}$$

If (A, B) is stabilizable and (A, C) is detectable, then the spectrum of the Hamiltonian matrix is $\sigma(H) = \{\pm\lambda_1, \ldots, \pm\lambda_n\}$, $\mathrm{Re}\,(\lambda_k) < 0$ for $k = 1, \ldots, n$. Moreover, if the columns of $\begin{bmatrix} U \\ V \end{bmatrix}$, $U, V \in \mathbb{R}^{n\times n}$, span the H-invariant subspace corresponding to $\lambda_1, \ldots, \lambda_n$, then the symmetric, positive semidefinite stabilizing solution of the ARE (7) is given by $P_* := -VU^{-1}$. Any method for solving nonsymmetric eigenproblems returning a basis for the required invariant subspace can be used to solve this problem. The most prominent methods using this connection to Hamiltonian matrices are the eigenvector approach and the Schur vector-type methods, which rely on similarity transformations $S^{-1}\,H\,S$ to obtain a reduced form from which the desired H-invariant subspace can be read off. Alternatively, spectral projection methods such as the sign function method can be employed. A discussion of these methods, as well as original references, can be found in [16] and [24]–[26]. The methods based on the sign function serve as the basis for parallel matrix algorithms for the LQR problem. We discuss their parallel implementation later. The first step in the computational process, which is a similarity transformation to Hessenberg or Hessenberg-like form or the first iteration in (4), destroys the sparsity structure of A and the low-rank nature of the off-diagonal blocks in (8). Therefore, these methods have a prohibitive cost of $\mathcal{O}(n^3)$ flops and cannot be applied to large-scale sparse problems.

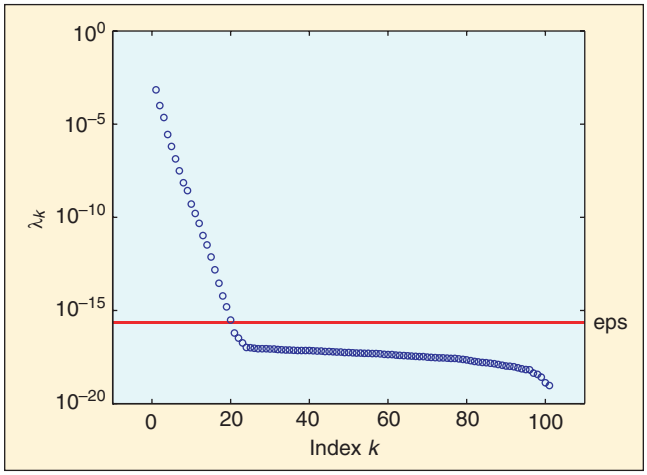

Figure 2. *Decay of eigenvalues of P_h in the stabilizing Riccati solution. The eigenvalues below the* eps*-line can be set to zero without introducing any significant error in the spectral decomposition of P_h. With increased dimension, the number of eigenvalues larger than machine precision (almost) does not increase.*

Large-scale eigenvalue problems are often solved using Krylov subspace methods; see [11]. These methods do not alter H but use it only for matrix-vector products, which can be implemented efficiently, employing the given sparsity and low-rank structure of H. Unfortunately, to obtain P_*, the full n-dimensional stable invariant subspace has to be computed, making the cost of computation and storage prohibitive. Although methods to compute low-rank approximations to the required invariant subspace have been proposed (see [27] and [28]), it is not clear whether the computed approximation $\tilde{P}$ to P_* gives a small residual $\|\mathcal{R}(\tilde{P})\|$. Moreover, $\tilde{P}$ usually does not have the required stabilizability property.

These considerations imply that methods based on the correspondence of the ARE (7) to the Hamiltonian matrix (8) are in general not suitable for solving large-scale LQR problems with sparse state matrix A.

The second class of methods consists of Newton-type methods that treat (7) as a nonlinear system of equations. In the following subsection, we use Newton's method to derive an algorithm for solving AREs by a sequence of linear systems of equations with the sparse matrix A. There is still the problem of storing the usually dense $n \times n$ matrix P_*. This storage can be avoided by observing that, for the problems under consideration, the spectrum of the positive semidefinite matrix $P_* = Z_* Z_*^T$ often decays to zero rapidly. Here Z_* can be considered as a Cholesky factor of P_*. A typical situation is given in Figure 2, where we have plotted the eigenvalues of P_* for an LQR problem arising from a finite-element discretization of a one-dimensional heat control problem.

For an eigenvalue decay as in Figure 2, we expect that P_* can be approximated accurately by a factorization $\tilde{Z}\tilde{Z}^T$ for some $\tilde{Z} \in \mathbb{R}^{n\times r}$ with $r \ll n$. Such an approximation is obtained by truncating the spectral decomposition $P_* = \sum_{j=1}^{n} \lambda_j z_j z_j^T$ after the first r terms. Here, the eigenvalues λ_j are ordered by decreasing magnitude and z_j is the eigenvector of P_* corresponding to λ_j. This observation has led to various approaches for solving AREs (and Lyapunov equations) by methods based on low-rank factorization of the solution; see [23], [29], and [30].

So far, there are some partial results explaining the decay of the eigenvalues of Lyapunov and Riccati solutions; bounds and estimates for the decay are given in [31] and [32]. Structural information of the underlying physical problem has not yet been incorporated into the analysis. Such information might shed more light on the existence of accurate low-rank approximations.

A Sparse Method for the LQR Problem

Since the ARE (7) is a nonlinear system of equations, it is natural to apply Newton's method to find its solutions. This approach has been investigated; details and further references can be found in [16], [19], [25], and [26]. To make the derivation more concise, we remove the output and control weighting matrices S and R by redefining $B := BR_C^{-1}$, $C := S_C^{-1}C$, where $S = S_C^T S_C$ and $R = R_C^T R_C$ are the Cholesky factorizations of S and R, respectively.

Observe that the (Frechét) derivative of $\mathcal{R}$ at P is given by the Lyapunov operator

$$\mathcal{R}_P' : Z \to \left(A - BB^T P\right) Z + Z \left(A - BB^T P\right)^T$$

and note that Newton's method for AREs is given by $N_k := \left(\mathcal{R}_{P_k}'\right)^{-1} \mathcal{R}(P_k)$, $X_{k+1} = X_k + N_k$. Then the Newton iteration for a given starting matrix P_0 can be written as follows:

1) $A_k \leftarrow A - BB^T P_k$
2) Solve the Lyapunov equation $A_k^T N_k + N_k A_k = -\mathcal{R}(P_k)$
3) $P_{k+1} \leftarrow P_k + N_k$.

Assume a stabilizing P_0 is given such that A_0 is stable. Then all A_k are stable and the iterates P_k converge to P_* quadratically. To make this iteration work for large-scale problems, we need a Lyapunov equation solver that employs the structure of A_k as "sparse + low-rank perturbation" by avoiding to form A_k explicitly, and which computes a low-rank approximation to the solution of the Lyapunov equation following the low-rank factorization paradigm explained above. A relevant method is derived in detail in [29] and [33] and is described in the following.

First, we rewrite Newton's method for AREs such that the next iterate is computed directly from the Lyapunov equation in Step 2:

$$A_k^T P_{k+1} + P_{k+1} A_k = -C^T C - P_k BB^T P_k =: -W_k W_k^T. \quad (9)$$

Assuming that $P_k = Z_k Z_k^T$ for rank $(Z_k) \ll n$ and observing that rank $(W_k) \le m + p \ll n$, we need only a numerical method to solve Lyapunov equations having a low-rank right hand side which returns a low-rank approximation to the (Cholesky) factor of its solution. For this purpose, we can use a modified version of the alternating directions implicit (ADI) method for Lyapunov equations of the form $FQ + QF^T = -WW^T$ with F stable, $W \in \mathbb{R}^{n \times n_w}$. The ADI iteration can be written as [34]

$$\begin{aligned} \left(F^T + p_j I\right) Q_{(j-1)/2} &= -WW^T - Q_{j-1}(F - p_j I), \\ \left(F^T + \overline{p_j} I\right) Q_j^T &= -WW^T - Q_{(j-1)/2}(F - \overline{p_j} I), \end{aligned} \quad (10)$$

where $\overline{p}$ denotes the complex conjugate of $p \in \mathbb{C}$. If the shift parameters p_j are chosen appropriately, then $\lim_{j\to\infty} Q_j = Q$ with a superlinear convergence rate. Starting this iteration with $Q_0 = 0$ and observing that for stable F, Q is positive semidefinite, it follows that $Q_j = Y_j Y_j^T$ for some $Y_j \in \mathbb{R}^{n \times r_j}$. Inserting this factorization into the above iteration, rearranging terms and combining two iteration steps, we obtain a factored ADI iteration that in each iteration step yields n_w new columns of a full-rank factor of Q; see [29], [30], and [33] for several variants of this method. If convergence of the factored ADI iteration with respect to a suitable stopping criterion is achieved after $j_{\max}$ steps, then $Y_{j_{\max}} = [V_1, \ldots, V_{j_{\max}}] \in \mathbb{R}^{n \times j_{\max} n_w}$, where $V_j \in \mathbb{R}^{n \times n_w}$. For large n and small n_w we therefore expect that $r_{j_{\max}} := j_{\max} n_w \ll n$. In that case, we have computed a low-rank approximation $Y_{j_{\max}}$ to a factor Y of the solution, that is $Q = YY^T \approx Y_{j_{\max}} Y_{j_{\max}}^T$. In case $n_w \cdot j_{\max}$ becomes large, a column compression technique from [35] can be applied to reduce the number of columns in $Y_{j_{\max}}$ without adding significant error.

The term "low-rank approximation" is slightly misleading. If the ADI method runs to convergence, it computes an *exact* full-rank factor of the solution. Terminating based on a stopping criterion guaranteeing full attainable accuracy, we get an approximation as close to the original solution as can be expected from any numerical algorithm. That is, the ADI method can be used in the same way as Newton's method for AREs or for the computation of system Gramians as, for instance, the Bartels–Stewart method. Therefore, nonstabilizing Riccati solutions, which arise when using projection methods such as Krylov subspace methods, do not appear in this context.

For an implementation of this method, we need a strategy to select the shift parameters p_j. We do not discuss this problem here in detail; see [29] and [30] and the references therein for a detailed discussion. A numerically inexpensive, heuristic algorithm that gives good performance in practice can be found in [29]. Usually, a finite number of shifts is computed in advance and applied cyclically if the ADI method needs more iterations than the number of available shifts.

Since A_k is stable for all k we can apply the modified ADI iteration to (9). Then, $W = [C^T \;\; P_k B]$ and hence, $n_w = m + p$, so that usually $n_w \ll n$. Also note that the above algorithm can be implemented in real arithmetic by combining two steps, even if complex shifts need to be

used, which may be the case if A_k is nonsymmetric. A complexity analysis of the factored ADI method depends on the method used for solving the linear systems in each iteration step. If applied to $F = A_k^T$ from (9), we have to deal with the situation that A_k is a shifted sparse matrix plus a low-rank perturbation. If we can solve for the shifted linear system of equations in (10) efficiently, the low-rank perturbation can be dealt with using the Sherman–Morrison–Woodbury formula [11] in the following way: let k be the index of the Newton iterates and let j be the index of the ADI iterates used to solve the kth Lyapunov equation, respectively, and set $K_k := B^T P_k$. Then

$$\begin{aligned}\left(F^T + p_j^{(k)} I_n\right)^{-1} &= \left(A + p_j^{(k)} I_n - BK_k\right)^{-1} \\ &= \left(I_n + L_k (I_m - K_k L_k)^{-1} K_k\right)\left(A + p_j^{(k)} I_n\right)^{-1},\end{aligned}$$

where $L_k := \left(A + p_j^{(k)} I_n\right)^{-1} B$. Hence, all linear systems of equations to be solved in one iteration step have the same coefficient matrix $A + p_j^{(k)} I_n$. If $A + p_j^{(k)} I_n$ is a banded matrix or can be reordered to become banded, then a direct solver can be employed. If workspace permits, it is desirable to compute a factorization of $A + p_j^{(k)} I_n$ for each different shift parameter beforehand (usually, only a few parameters are used). These factorizations can then be used in each iteration step of the ADI iteration. In particular, if A is symmetric positive definite, as is the case in many PDE-constrained optimal control problems, and can be reordered as a narrow-banded matrix, then each factorization requires $\mathcal{O}(n)$ flops, and the total cost $\mathcal{O}(k_{\max} \max(j_{\max}) n)$ scales with n as desired. If iterative solvers are employed for the linear systems, it should be noted that only one Krylov space needs to be computed (see [30] for details) and hence we obtain an efficient variant of the factored ADI iteration. The same implementation of the factored ADI method for solving stable Lyapunov equations can also be used for balanced truncation of systems with sparse A efficiently; see the corresponding section below.

If the solution of the ARE (7) is only a detour to obtaining the optimal feedback matrix defining the optimal control u_* of the LQR problem, then we can avoid forming the Y_j's in the factored ADI iteration and directly update the approximation to the optimal feedback $K_* := B^T P_* = B^T Z_* Z_*^T$. Observe that in each Newton step, the approximation $K_k = B^T Z_k Z_k^T$ to $K_* = \lim_{k\to\infty} K_k$ is computed. Now Z_k is the converged factor of the ADI iteration and therefore $Z_k = [V_1, V_2, \ldots, V_{j_{\max}}]$. Thus

$$K_k = B^T Z_k Z_k^T = \sum_{j=1}^{j_{\max}} B^T V_j V_j^T.$$

Hence, K_k can be updated directly when a new V_j is computed: for $K_k^{(0)} = 0 \in \mathbb{R}^{m\times n}$,

$$K_k^{(j)} := K_k^{(j-1)} + \left(B^T V_j\right) V_j^T, \quad j = 1, \ldots, j_{\max}, \tag{11}$$

and $K_k := K_k^{(j_{\max})}$. For updating $K_k^{(j)}$ using the above formula, only a constant workspace of size $m \times n$ and $4nm^2$ flops are necessary.

The tools available in MATLAB and the algorithmically more advanced SLICOT library have limited ability to exploit sparse structures.

Thus, the solution of an LQR problem for a time-dependent linear PDE proceeds as follows: first, determine a semidiscretization of the problem, resulting in a standard LTI system (1). Then compute the optimal feedback matrix K_* as $\lim_{k\to\infty} \lim_{j\to\infty} K_k^{(j)}$ using Newton's method for AREs, with the factored ADI iteration for solving the Lyapunov equations in each step, and the direct feedback updating as in (11).

There exists a thorough convergence theory for solving LQR problems with the above approach. It can be shown that the finite-dimensional regulators converge uniformly to the infinite-dimensional regulator if an appropriate framework for the semidiscretization is chosen. This framework covers standard Ritz–Galerkin type discretizations underlying the finite element method. Note that using FEM, we obtain an LTI system

$$M\dot{x} = -Sx + Bu, \quad y = Cx, \tag{12}$$

where M is the positive definite mass matrix and S is the stiffness matrix corresponding to the FEM applied to the spatial differential operator. In many situations, S is symmetric positive definite and hence, $A := -M^{-1}S$ is stable. But of course, we prefer to avoid forming A explicitly since A is usually a dense matrix. There are several approaches that can be used to treat this problem. For instance, to solve a linear system of the form $(A + pI)v = b$ during the ADI iteration, the coefficient matrix has the form $-M^{-1}S + pI_n = -M^{-1}(S - pM)$. Thus, at the additional cost of a sparse matrix-vector multiply, we obtain the solution v from the linear system of equations $(S - pM)v = -Mb$. Note that for $p < 0$, $S - pM$ is positive definite if M is positive definite even if S is only positive semidefinite.

The factored ADI iteration for Lyapunov equations and Newton's method based on the factored ADI iteration are implemented in the MATLAB-based package LYAPACK [36], available from http://www.netlib.org/lyapack. It allows the solution of large, sparse LQR problems in MATLAB.

An Application: Optimal Cooling of Steel Profiles

We consider the problem of optimal cooling of steel profiles. This problem arises in a rolling mill when different steps in the production process require different temperatures of the raw material. To achieve a high production rate, it is necessary to reduce the temperature as fast as possible to the required level before entering the next production phase. At the same time, the cooling process, which is realized by spraying cooling fluids on the surface, has to be controlled so that material properties, such as durability or porosity, satisfy given quality standards. Large gradients in the temperature distributions of the steel profile may lead to unwanted deformations, brittleness, loss of rigidity, and other undesirable material properties.

The problem is modeled using a boundary control given by the temperature of the cooling fluid for a heat-diffusion process described by the linearized heat equation. It is reasonable to assume that the steel profile has infinite length. Therefore, diffusion in the z-direction is negligible, and we can work with a 2-D computational domain given by a vertical cut through the profile. Moreover, due to symmetry of the 2-D domain, we can halve the computational domain to obtain Ω as in [37]; see Figure 3. The boundary is partitioned into eight pieces $\Gamma = \bigcup_{k=1,\dots,8} \Gamma_k$, where the symmetry is modeled by means of a homogeneous Neumann boundary for Γ_8. For each of the other parts of the boundary, there is one spraying nozzle that cools the part uniformly. As an idealized situation we assume constant heat conductivity λ, heat capacity c, and density ρ. Variations in these parameters are marginal for a temperature range in which no phase transition of the non-solid steel occurs. In this idealized situation, the modeling equations are

$$\frac{\partial}{\partial t}\mathbf{x} = \frac{\lambda}{c \cdot \rho}\Delta\mathbf{x} \quad \text{in } \Omega, \tag{13}$$

$$\frac{\partial}{\partial n}\mathbf{x} = \frac{1}{\lambda}(u_k - \mathbf{x}), \xi \in \Gamma_k, \ k = 1, \dots, 7, \tag{14}$$

$$\frac{\partial}{\partial n}\mathbf{x} = 0, \qquad \xi \in \Gamma_8. \tag{15}$$

Here, $\mathbf{x} = \mathbf{x}(t; \xi, \eta)$ denotes the temperature in $(\xi, \eta) \in \bar{\Omega} = \Omega \cup \Gamma$ at time t, $u_k(t)$ is the temperature of the cooling fluid at time t, and $\partial/\partial n$ denotes the normal derivative. The initial temperature profile is shown in Figure 3.

The spatial discretization is obtained using the FEM implementation ALBERT [38] with linear Lagrange elements. The initial mesh, obtained by Delaunay triangulation and leading to a state-space dimension with $n = 106$, is shown in Figure 3. Using a global refinement strategy, based on one, two, or three steps of bisection, leads to linear systems of the form (12) with orders $n = 371$, 1357, 5177. The twice-refined grid is shown in Figure 4.

The stiffness and mass matrices K and M have the sparsity pattern shown in the left-hand plot of Figure 5, the right-hand plot displays the sparsity pattern of their Cholesky factors. (Here, nz is the number of nonzeros in the matrix.) Solving the linear systems occurring in the ADI

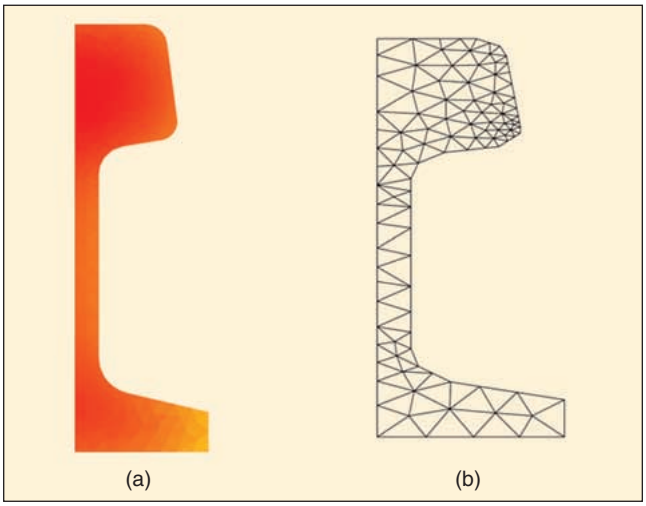

Figure 3. *(a) Initial temperature distribution and (b) computational domain and initial FE mesh for the steel cooling example. The initial temperature distribution was computed using the uncontrolled heat equation, simulating cooling at air. The computational domain was generated using MATLAB's PDE Toolbox, based on geometry data from [37].*

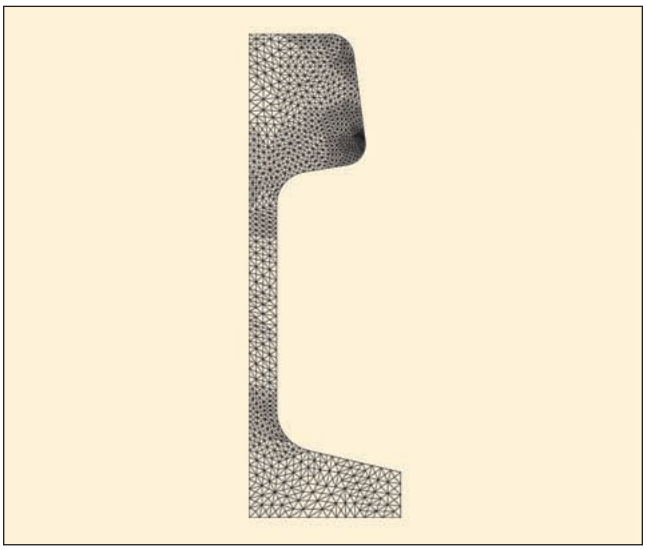

Figure 4. *Globally refined FE mesh for the steel cooling example. The figure shows the mesh after two refinement-by-bisection steps by MATLAB's PDE Toolbox.*

iteration with direct methods leads to a severe fill-in and a complexity of $\mathcal{O}(n^3)$ flops. Applying the symmetric reverse Cuthill–McKee algorithm for bandwidth reduction of sparse matrices yields the sparsity patterns of K, M, and the Cholesky factor of M in Figure 6. This pattern reduces the CPU time needed to compute the Cholesky factor from 8.3 to 0.6 s on a Linux PC (AMD Athlon processor, running at 1.7 GHz) using MATLAB 6.5. Now the solution of the resulting narrow-banded linear systems can essentially be achieved by direct methods in $\mathcal{O}(n)$ flops. Hence, the computational work for the solution of the LQR problem by means of the ADI-Newton method scales with n.

Figure 7 shows the evolution of the temperature distribution in the steel profile using optimal LQ feedback control. As can be seen, cooling with acceptable temperature gradients is achieved.

Parallel Methods for LQR Problems

As discussed in the subsection on parallel matrix algorithms, it is highly attractive to consider the sign function

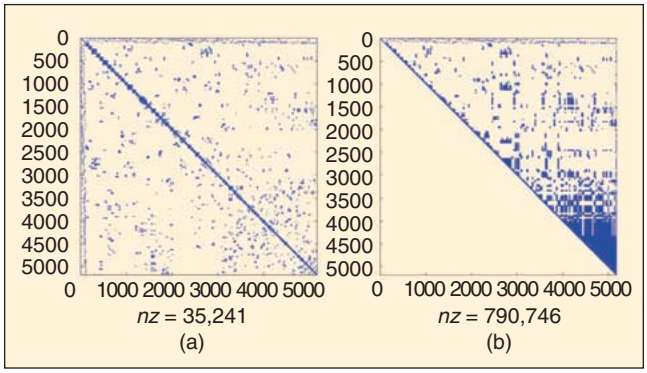

Figure 5. *Sparsity pattern of (a) mass and stiffness matrix and (b) Cholesky factor of mass matrix of the steel cooling example (n = 5,117). The Cholesky factorization leads to severe fill in, making the solution of linear systems of equations almost as expensive as for a dense coefficient matrix.*

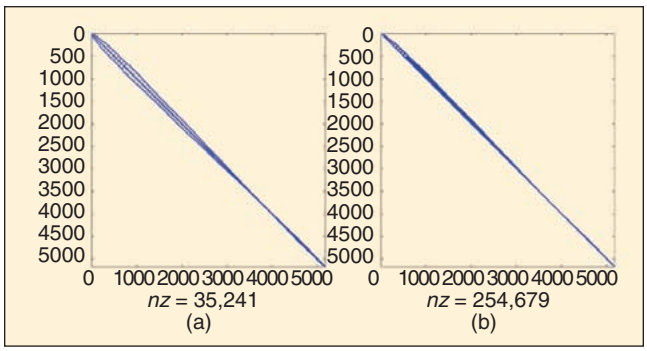

Figure 6. *Sparsity pattern of (a) mass and stiffness matrix and (b) Cholesky factor of mass matrix of the steel cooling example (n = 5,117) after reverse Cuthill–McKee permutation. The achieved band structure allows for the cheap solution of linear systems of equations using band solvers.*

method as an essential tool for solving large-scale LQR problems on parallel computers. The basic approach to solving the LQR problem (as well as the LQG and the H_∞ optimal control problems) is to apply the sign function method to the Hamiltonian matrix H from (8). Note that the sign function also has the advantage of being structure-preserving: since H^{-1} and $H + H^{-1}$ are also Hamiltonian, the iteration (4) preserves the problem structure. As mentioned before, from sign(H) we can obtain a spectral projector onto the required stable H-invariant subspace. An orthogonal basis could be computed from a (rank-revealing) QR decomposition of $I_{2n} - \text{sign}(H)$, and the solution to the ARE (7) would be obtained as in the Schur vector method by computing $X = -VU^{-1}$ if the columns of $\left[\begin{smallmatrix} U \\ V \end{smallmatrix}\right]$ span this subspace. This step is usually avoided by observing that i) the columns of $\left[\begin{smallmatrix} I_n \\ -X \end{smallmatrix}\right]$ also form a basis of the stable H-invariant subspace, and ii) $\mathcal{P}_+ := \frac{1}{2}(I_{2n} + \text{sign}(H))$ is a projector onto the antistable H-invariant subspace. Thus, the columns of $\left[\begin{smallmatrix} I_n \\ -X \end{smallmatrix}\right]$ are in the nullspace of $\mathcal{P}_+$ and hence, with

$$\text{sign}(H) = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix},$$

we obtain

$$0 = \mathcal{P}_+ \begin{bmatrix} I_n \\ -X \end{bmatrix} = \frac{1}{2} \begin{bmatrix} I_n + S_{11} & S_{12} \\ S_{21} & I_n + S_{22} \end{bmatrix} \begin{bmatrix} I_n \\ -X \end{bmatrix}.$$

This identity yields the overdetermined but consistent linear system of equations

$$\begin{bmatrix} S_{12} \\ I_n + S_{22} \end{bmatrix} X = \begin{bmatrix} I_n + S_{11} \\ S_{21} \end{bmatrix}, \tag{16}$$

which can be solved using a QR decomposition. Thus, the solution of the ARE (7) and the LQR problem is obtained by applying the sign function method to the corresponding Hamiltonian matrix H and solving (16). Both steps consist of parallelizable computations available in ScaLAPACK. In [39] an efficient implementation of this approach is discussed.

The potential of the sign function method for parallel computing was already recognized in [40]. Early attempts suffered from the fact that the parallelizations were not portable but rather were targeted to specific parallel computing platforms since at that time, ScaLAPACK and other means for enhancing the portability of parallel codes were not available. Moreover, the target platforms were usually supercomputers not available to most users, while the new parallelization effort only requires PCs or workstations connected in a LAN on the hardware side. By using the sign function approach on a commodity PC cluster consisting of eight PCs with 512 MB of main memory, we can solve problems of order $n = 6,000$. If 24 such systems are available in a LAN, problems of order $n = 10,000$ can be handled.

Without giving further details, we note that the spectral projection approach can also be used to avoid the solu-

tion of the ARE completely by computing the optimal feedback matrix directly. This approach is particularly appealing if the control weighting matrix R is ill-conditioned. Details of this approach and its parallelization, as well as performance results showing the almost perfect speed-up of spectral projection methods applied to the LQR problem, are given in [39].

Model Reduction

The model reduction problem is to find a reduced-order LTI system

$$\begin{aligned} \dot{\tilde{x}}(t) &= \tilde{A}\tilde{x}(t) + \tilde{B}u(t), \quad \tilde{x}(0) = \tilde{x}^0, \\ \tilde{y}(t) &= \tilde{C}\tilde{x}(t) + \tilde{D}u(t), \end{aligned} \tag{17}$$

of order $r \ll n$, such that, for the same input function u, the output $\tilde{y}$ is a good approximation to the output y of the original system (1). Hereafter, we assume that A is a stable matrix. Hence, the model reduction should also yield a stable matrix $\tilde{A}$ and stable transfer function $\tilde{G}(s) := \tilde{C}(sI - \tilde{A})^{-1}\tilde{B} + \tilde{D}$. Taking Laplace transforms and assuming H_2 inputs, the stability of G and $\tilde{G}$ implies that the outputs are H_2 signals. Assuming $x_0 = 0$ and $\tilde{x}_0$, we can bound the error in state-space, as well as in frequency domain, by the approximation error in the transfer function:

$$\|y - \tilde{y}\|_{L_2[0,\infty)} = \|y - \tilde{y}\|_{H_2} \le \left\|G - \tilde{G}\right\|_\infty \|u\|_{H_2}, \tag{18}$$

where $\|G\|_\infty := \operatorname{ess\,sup}_{\omega\in\mathbb{R}} \sigma_{\max}(G(\mathcal{J}\omega))$ is the H_∞ norm of G. Thus, many model reduction methods for control systems design aim at minimizing $\|G - \tilde{G}\|_\infty$, although for a given r, finding $\tilde{G}$ that minimizes $\|G - \tilde{G}\|_\infty$ is an open problem even in the scalar case [41]. Alternative minimization objectives possibly depending on the design problem to be solved include frequency-weighted model and controller reduction. A comprehensive introduction to model reduction methods and references to original work can be found in [42] and [43].

We focus here on minimizing the absolute error $\|G - \tilde{G}\|_\infty$, in particular on model reduction based on balanced truncation. The proposed method can be viewed as a prototype for applying alternative balancing-related model reduction techniques to large-scale problems.

Balanced truncation belongs to the class of model reduction methods that rely on truncating state-space transformations defined by means of a nonsingular matrix $T \in \mathbb{R}^{n\times n}$, so that

$$TAT^{-1} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad TB = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}, \quad CT^{-1} = [C_1 \quad C_2],$$

where $A_{11} \in \mathbb{R}^{r\times r}$, and TB and CT^{-1} are conformably partitioned. With $T = [T_l^T, L_l^T]^T \in \mathbb{R}^{n\times n}$ and $T^{-1} = [T_r, L_r]$, $T_l \in \mathbb{R}^{r\times n}$, $T_r \in \mathbb{R}^{n\times r}$, the reduced-order model is given by the projections

$$\begin{aligned} \tilde{A} &:= T_l A T_r = A_{11}, \quad \tilde{B} := T_l B = B_1, \\ \tilde{C} &:= C T_r = C_1, \quad \tilde{D} := D. \end{aligned} \tag{19}$$

For given r, the problem now is to find T_l, T_r such that $\|G - \tilde{G}\|_\infty$ is small.

The most common approach to truncation-based model reduction involves balancing the controllability Gramian W_c and the observability Gramian W_o, which solve the Lyapunov equations

$$AW_c + W_c A^T + BB^T = 0, \quad A^T W_o + W_o A + C^T C = 0. \tag{20}$$

Since W_c and W_o are positive semidefinite, they can be factored as $W_c = S^T S$ and $W_o = R^T R$. When the factors $S, R \in \mathbb{R}^{n\times n}$ are chosen to be triangular, they are the Cholesky factors of the Gramians.

From a numerical point of view, the observation that a balanced truncation approximation can be achieved using the product SR^T, instead of the product of the Gramians themselves, is a key ingredient of a reliable implementation of balanced model reduction. The resulting square-root (SR) algorithms avoid working with the Gramians

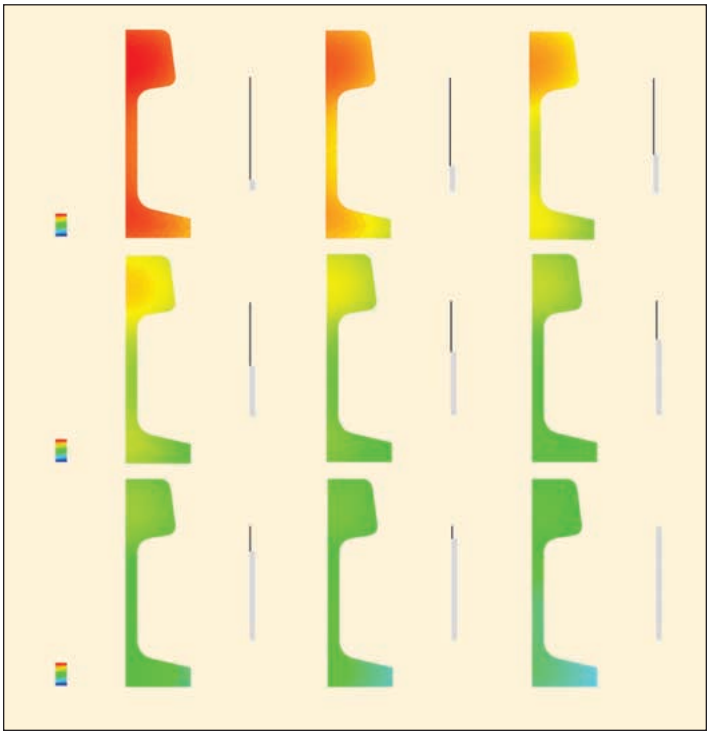

Figure 7. *Temperature in the optimally cooled profile. The color bars to the left give the color map for the temperature (1,000–500 °C), and the bar to the right of each plot shows the progress in time.*

since their condition number is the square of the condition numbers of the Cholesky factors. In these algorithms, (20) is initially solved for the Cholesky factors without forming the Gramians explicitly. The Cholesky factor computation can be achieved, for example, by Hammarling's method; see [16] and references therein, or an algorithm described in [23]. Then the singular value decomposition (SVD)

$$SR^T = [U_1\ U_2]\begin{bmatrix}\Sigma_1 & 0\\ 0 & \Sigma_2\end{bmatrix}\begin{bmatrix}V_1^T\\ V_2^T\end{bmatrix},$$
$$\Sigma_1 = \operatorname{diag}(\sigma_1,\ldots,\sigma_r),\quad \Sigma_2 = \operatorname{diag}(\sigma_{r+1},\ldots,\sigma_n) \quad (21)$$

is computed, where

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} \geq \sigma_{r+2} \geq \cdots \geq \sigma_n \geq 0. \quad (22)$$

If $\sigma_r > 0$ and $\sigma_{r+1} = 0$, that is, $\Sigma_2 = 0$, then r is the McMillan degree of the given LTI system, that is, the order of a minimal realization of (1). For a successful model reduction, r should be chosen to give a natural separation of the states, that is, one should search for a large gap $\sigma_r \gg \sigma_{r+1}$.

Finally, the matrices T_l and T_r yielding the reduced-order model (19) for the balancing state-space transformation are determined by

$$T_l = \Sigma_1^{-1/2} V_1^T R \quad \text{and} \quad T_r = S^T U_1 \Sigma_1^{-1/2}. \quad (23)$$

It is known that, for every choice of r, such that $\sigma_r > \sigma_{r+1}$ in (21), yields a stable, minimal, and balanced reduced model. The Gramians corresponding to the resulting transfer function $\tilde{G}(s)$ are both equal to Σ_1. Detailed discussions of balanced truncation and the square-root methods for implementing them can be found in [42]–[44].

Serial implementations of balanced truncation algorithms and other balancing-related model reduction techniques are described in [44]. The described implementations are contained in SLICOT [2], available at http://www.win.tue.nl/niconet/NIC2/slicot.html.

Though balanced truncation does not generally yield the best rth order approximant of G in the H_∞ norm, we obtain the error bound

$$\left\| G - \tilde{G} \right\|_\infty \leq 2 \sum_{k=r+1}^{n} \sigma_k, \quad (24)$$

which is attainable. This a priori bound makes balanced truncation attractive since it allows an adaptive choice of the order r of $\tilde{G}$. Because of this error bound, it is desirable to apply balanced truncation to large-scale models. However, Schur vector solutions of the Lyapunov equations requires $\mathcal{O}(n^3)$ flops and $\mathcal{O}(n^2)$ workspace. Even if these requirements could be reduced, the SVD in (21) requires $O(n^3)$ flops and $\mathcal{O}(n^2)$ workspace. So for the moment, we focus on reducing the required resources for this computational step by employing low-rank factorizations. This approach turns out to be the key to the success of the sparse and parallel model reduction algorithms.

The basic idea is to replace the Cholesky factors of the Gramians with low-rank factors, resulting in a smaller arithmetic cost and workspace requirement. So far, we have assumed that the Cholesky factors S and R of the Gramians are square $n \times n$ matrices. For nonminimal systems, we have rank $(S) < n$ and/or rank $(R) < n$. Hence, rather than working with the singular Cholesky factors, we may use full-rank factors of W_c, W_o. Since W_c and W_o are positive semidefinite, there exist matrices $\hat{S} \in \mathbb{R}^{n_c \times n}$, $\hat{R} \in \mathbb{R}^{n_o \times n}$, such that $W_c = \hat{S}^T\hat{S}$, $W_o = \hat{R}^T\hat{R}$, and

$$n_c := \operatorname{rank}(\hat{S}) = \operatorname{rank}(S) = \operatorname{rank}(W_c),$$
$$n_o := \operatorname{rank}(\hat{R}) = \operatorname{rank}(R) = \operatorname{rank}(W_o).$$

Although the full-rank factors $\hat{S}$, $\hat{R}$ can in principle be obtained from S and R, it is more efficient to compute $\hat{S}$ and $\hat{R}$ directly. In the latter case, we have $S := \begin{bmatrix}\hat{S}\\ 0\end{bmatrix}$, $R := \begin{bmatrix}\hat{R}\\ 0\end{bmatrix}$. The SVD in (21) can then be obtained from that of $\hat{S}\hat{R}^T$ as follows. Here we assume $n_c \geq n_o$; the case $n_c < n_o$ can be treated analogously. First we compute the SVD

$$\hat{S}\hat{R}^T = \hat{U}\begin{bmatrix}\hat{\Sigma}\\ 0\end{bmatrix}\hat{V}^T, \quad \hat{\Sigma} = \operatorname{diag}(\sigma_1,\ldots,\sigma_{n_o}), \quad (25)$$

where $\hat{U} \in \mathbb{R}^{n_c \times n_c}$, $\hat{V} \in \mathbb{R}^{n_o \times n_o}$. Partitioning $\hat{U} = [\hat{U}_1 \quad \hat{U}_2]$ such that $\hat{U}_1 \in \mathbb{R}^{n_c \times n_o}$, the SVD of SR^T is given by

$$SR^T = \begin{bmatrix}\hat{U}_1 & \hat{U}_2 & 0\\ 0 & 0 & I_{n-n_c}\end{bmatrix}\begin{bmatrix}\hat{\Sigma} & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}\hat{V}^T & 0\\ 0 & I_{n-n_o}\end{bmatrix}. \quad (26)$$

The choice of r yielding the McMillan degree of the system or the size of the reduced order model can be based on the singular values of $\hat{\Sigma}$. Note that the subsequent computations can be performed working only with $\hat{U}_1$, $\hat{\Sigma}$, and $\hat{V}$ rather than using the data from the full-size SVD in (26). This technique yields a significant savings in workspace and computational cost. Using complexity estimates from [11], (21) requires $22n^3$ flops and workspace for $2n^2$ real numbers if U, V are formed explicitly, whereas (25) requires only $14n_cn_o^2 + 8n_o^3$ flops and workspace for $n_c^2 + n_o^2$ real numbers. In practice, for large-scale dynamical systems, the numerical rank of W_c, W_o and $\hat{S}$, $\hat{R}$ is often much smaller than n; see [32], [45]. Suppose that $n_c = n_o = n/100$, which is typical for the semi-discretization of parabolic PDEs. Then (25) is 1 million times less expensive than (21) and only 0.01% of the workspace is required for (25) as compared to (21). More savings are obtained from the cheaper computation of the truncation matrices yielding the reduced-order models.

Full-rank factorizations and their low-rank approximations can be used for efficient model reduction for sparse systems and for large-scale dense systems using parallel

computing. These techniques are described in the following two subsections.

Sparse Balanced Truncation

Model reduction of large, sparse systems has been widely considered because of its relevance to circuit simulation for validating VLSI layouts; see [13]. These methods are mainly based on partial realization (matching the leading Markov parameters of the system) or Padé approximations (matching the leading moments of the system). The underlying computational tools are Krylov space methods based on either the Lanczos or Arnoldi processes such as in Padé via Lanczos (PVL); see [10], [13], and [42].

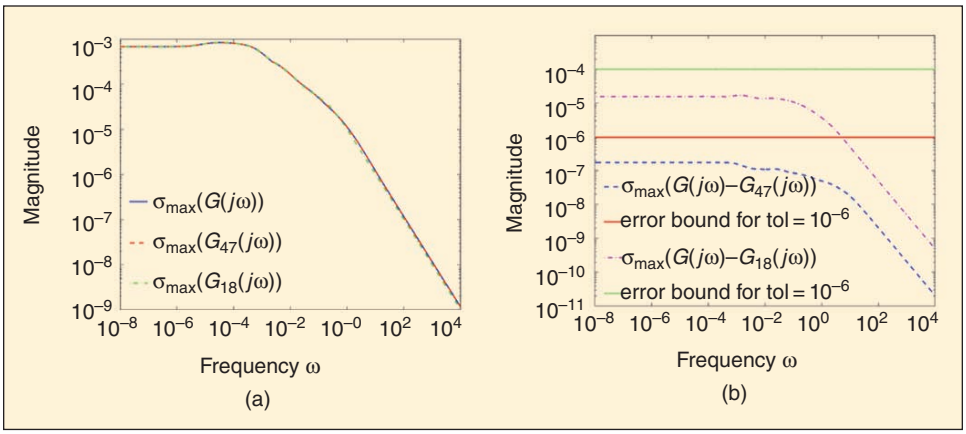

Figure 8. *Accuracy of reduced order models ($r = 47$ and $r = 18$) for the steel cooling example ($n = 5,117$) computed using sparse balanced truncation, (a) Sigma plot for original and reduced system and (b) absolute error in $\| \ \|_\infty$-norm. The reduced-order models clearly satisfy the error bound (24).*

Unfortunately, these methods do not necessarily give a stable reduced-order model, the selection of r is not automatic since there is no computable error bound as in (24), and the choice of expansion points in the more advanced rational Krylov-type methods is problematic. Therefore, attempts have been made to apply balanced truncation to sparse systems. The basic observation is that by using the low-rank factorization paradigm, we can perform balanced model reduction by efficiently solving sparse Lyapunov equations. This idea is pursued in [29] and [30], leading to the development of the factored ADI iteration described in the LQR section. Using this method to solve the Lyapunov equations in (20) for the factors of the Gramians, the cheap singular value decomposition (25) can be employed, yielding a balanced truncation method for sparse systems.

We have applied an ADI-based balanced truncation method to obtain a reduced-order approximation for the $n = 5177$ model from the steel cooling example described above employing a modified version of the MATLAB function `lp_lrsrm` from LYAPACK [36], which can be used to compute reduced-order models of fairly large systems even within MATLAB. Factors of the Gramians were computed using the implementation of the ADI method provided by LYAPACK. The computed factors $\hat{S}$ and $\hat{R}$ have 441 and 438 rows. The reduced-order model was selected to satisfy $\|G - \tilde{G}\|_\infty < 10^{-6}$ based on (24). Thus the order of $\tilde{G}$ was computed to be $r = 47$, and the actual value of the bound (24) is $9.67 \cdot 10^{-7}$. Note that the model error resulting from the finite-element semidiscretization is $\mathcal{O}(h^2) \equiv \mathcal{O}(10^{-4})$ so that the contribution of the truncation error from model reduction with an error tolerance of 10^{-6} is negligible when doing simulations or control design with the resulting low-dimensional model. If we require an error of the same order as the discretization error, that is, $\|G - \tilde{G}\|_\infty < 10^{-4}$, then a reduced-order model of size $r = 18$ is sufficient. In that case, evaluating the error bound (24) yields $9.99 \cdot 10^{-5}$.

Figure 8 shows the singular value ("sigma") plot of the original and reduced-order systems as well as the resulting absolute error.

Parallel Model Reduction

The approach to an implementation of balanced truncation on parallel computers is described in detail in [45]. The main idea is to follow the low-rank factorization paradigm outlined above, and compute the low-rank factors of the system Gramians using a sign-function based method to obtain an efficient parallel algorithm.

Roberts [18] was the first to use the matrix sign function for solving Lyapunov (and Riccati) equations. In the proposed method, the solution of the stable Lyapunov equation

$$A^T X + XA + Q = 0 \tag{27}$$

is computed by applying the Newton iteration (4) to the Hamiltonian matrix $H = \begin{bmatrix} A & 0 \\ Q & -A^T \end{bmatrix}$ corresponding to (27). The solution X_* can then be obtained directly from $\operatorname{sign}(H) = \begin{bmatrix} -I & 0 \\ 2X_* & I \end{bmatrix}$.

Iteration (4) can be used to compute the observability Gramian directly by setting $Q := C^T C$ in (27). For the controllability Gramian, we replace A by A^T while setting $Q := BB^T$. Combining both iterations so that W_c and W_o are computed simultaneously and observing that (4) applied to $\begin{bmatrix} A & 0 \\ Q & -A^T \end{bmatrix}$ yields decoupled iterations on the blocks of the matrix, we obtain the following algorithm for $A_0 := A$, $P_0 := BB^T$, $Q_0 := C^T C$.

FOR $k = 0, 1, 2, \ldots$ *until convergence*

$$A_{k+1} \leftarrow \frac{1}{2c_k}\left(A_k + c_k^2 A_k^{-1}\right),$$
$$P_{k+1} \leftarrow \frac{1}{2c_k}\left(P_k + c_k^2 A_k^{-1} P_k \left(A_k^{-1}\right)^T\right),$$
$$Q_{k+1} \leftarrow \frac{1}{2c_k}\left(Q_k + c_k^2 \left(A_k^{-1}\right)^T Q_k A_k^{-1}\right). \qquad (28)$$

At convergence, we obtain $W_c = (\lim_{k\to\infty} P_k)/2$, and $W_o = (\lim_{k\to\infty} Q_k)/2$.

Iteration (28) can be modified to obtain factors rather than the solutions themselves. The basic idea is that the iterations for the symmetric positive definite matrices P_k, Q_k can be written in factored form as $P_{k+1} = B_{k+1}B_{k+1}^T$, $Q_{k+1} = C_{k+1}^T C_{k+1}$, where $B_0 := B$, $C_0 := C$. To compute full-rank factors of the Gramians, we must not allow the iterates C_k, B_k to become rank deficient. This requirement can be enforced by using column or row compression techniques based on rank-revealing QR or LQ factorizations. The iterates thus obtained converge to the full-rank factors of the Gramians. Details of this procedure can be found in [23] and [45].

Numerical results and parallel performance of the resulting method and related model reduction methods are contained in [46]. The parallel efficiency is almost optimal so that truly large-scale problems can be tackled. However, if the original model has sparse matrix structure, then the sparse methods described in the last subsection are more appropriate. For large-scale problems with $n > 10^5$ it is necessary to combine sparse algorithms and parallel computing techniques. For instance, preliminary results suggest that parallel implementation of sparse balanced truncation can successfully be applied to models of order $n = 200,000$.

Conclusions

In this article we discussed approaches to solving large-scale control problems. It has been shown that using advanced techniques from numerical linear algebra and modern matrix algorithms, important analysis and synthesis problems in systems and control theory can be solved using traditional methods. Therefore, the "curse of dimensionality" should not prevent us from modeling control systems at a high level of sophistication. The necessary computational tools can be run on desktop computers or using local area network-based parallel computing techniques that can be installed without ingenious computer science skills.

Some of the techniques also carry over to the discrete-time case. While the development of feasible sparse algorithms for some of the major computational problems arising in discrete-time control problems such as the Stein (discrete Lyapunov) and discrete-time algebraic Riccati equations is still under development, parallel algorithms for model reduction and the solution of the LQR problem for discrete-time systems are described in [39] and [46].

Acknowledgments

I would like to thankfully acknowledge the work of my student Jens Saak for producing the figures related to the optimal cooling of steel profiles. Moreover, I am grateful to my deceased colleague and friend Thilo Penzl, whose ideas inspired much of the work related to solving large-scale sparse matrix equations. I would also like to the work of Ralph Byers, Heike Fassbender, Rafael Mayo, and Enrique and Gregorio Quintana-Ortí. Only through our collaboration over many years could the results presented in this article be achieved. This work was supported by the DFG Research Center "Mathematics for key technologies" (FZT 86) in Berlin. The comments of four referees greatly helped to improve this manuscript. Their effort is gratefully acknowledged.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide,* 3rd ed. Philadelphia, PA: SIAM, 1999.

[2] P. Benner, V. Mehrmann, V. Sima, S. Van Huffel, and A. Varga, "SLICOT—A subroutine library in systems and control theory," in *Applied and Computational Control, Signals, and Circuits,* B.N. Datta, Ed. Boston, MA: Birkhäuser, 1999, vol. 1, ch. 10, pp. 499—539.

[3] The MathWorks, Inc., http://www.matlab.com

[4] J.L. Lions, *Some Aspects of the Optimal Control of Distributed Parameter Systems* (Regional Conf. Ser. Appl. Math.). Philadelphia, PA: SIAM, 1972.

[5] H.T. Banks and K. Kunisch, "The linear regulator problem for parabolic systems," *SIAM J. Cont. Optim.*, vol. 22, no. 5, pp. 684–698, 1984.

[6] R.F. Curtain and H. Zwart, *An Introduction to Infinite-Dimensional Linear Systems Theory* (Texts in Applied Mathematics, vol. 21). New York: Springer-Verlag, 1995.

[7] I. Lasiecka and R. Triggiani, *Control Theory for Partial Differential Equations: Continuous and Approximation Theories I. Abstract Parabolic Systems*. Cambridge, UK: Cambridge Univ. Press, 2000.

[8] W. Hackbusch, "A sparse matrix arithmetic based on $\mathcal{H}$-matrices. I. Introduction to $\mathcal{H}$-matrices," *Computing,* vol. 62, no. 2, pp. 89–108, 1999.

[9] J. Saak, "Effiziente numerische Lösung eines Optimalsteuerungsproblems für die Abkühlung von stahlprofilen," Diplomarbeit, Fachbereich 3/Mathematik und Informatik, Universität Bremen, D-28334 Bremen, Sept. 2003.

[10] Z. Bai, "Krylov subspace techniques for reduced-order modeling of large-scale dynamical systems," *Appl. Numer. Math.,* vol. 43, no. 1/2, pp. 9–44, 2002.

[11] G.H. Golub and C.F. Van Loan, *Matrix Computations* 3rd ed. Baltimore, MD: Johns Hopkins Univ. Press, 1996.

[12] I.S. Duff, A.M. Erisman, and J.K. Reid, *Direct Methods for Sparse Matrices*, Oxford, UK: Oxford Science, 1993.

[13] R. Freund, "Reduced-order modeling techniques based on Krylov subspaces and their use in circuit simulation," in *Applied and Computational Control, Signals, and Circuits,* B.N. Datta, Ed. Boston, MA: Birkhäuser, 1999, vol. 1, ch. 9, pp. 435–498.

[14] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* Cambridge, MA: MIT Press, 1994.

[15] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley, *ScaLAPACK Users' Guide.* Philadelphia, PA: SIAM, 1997.

[16] V. Sima, *Algorithms for Linear-Quadratic Optimization* (Pure and Applied Mathematics, vol. 200). New York,: Marcel Dekker, 1996.

[17] G. Henry and R.A. van de Geijn, "Parallelizing the QR algorithm for the unsymmetric algebraic eigenvalue problem: Myths and reality," *SIAM J. Sci. Comput.,* vol. 17, no. 4, pp. 870–883, 1996.

[18] J.D. Roberts, "Linear model reduction and solution of the algebraic Riccati equation by use of the sign function," *Int.. J. Contr.* vol. 32, pp. 677–687, 1980.

[19] P. Lancaster and L. Rodman, *The Algebraic Riccati Equation*. Oxford: Oxford Univ. Press, 1995.

[20] C. Kenney and A.J. Laub, "The matrix sign function," *IEEE Trans. Automat. Contr.*, vol. 40, no. 8, pp. 1330–1348, 1995.

[21] Z. Bai, J. Demmel, J. Dongarra, A. Petitet, H. Robinson, and K. Stanley, "The spectral decomposition of nonsymmetric matrices on distributed memory parallel computers," *SIAM J. Sci. Comput.*, vol. 18, no. 5, pp. 1446–1461, 1997.

[22] M.M. Konstantinov, V. Mehrmann, and P.H. Petkov, "Perturbation analysis for the Hamiltonian Schur form," *SIAM J. Matrix Anal. Appl.*, vol. 23, no. 2, pp. 387–424, 2001.

[23] P. Benner and E.S. Quintana-Ortí, "Solving stable generalized Lyapunov equations with the matrix sign function," *Numer. Algorithms,* vol. 20, no. 1, pp. 75–100, 1999.

[24] P. Benner, "Computational methods for linear-quadratic optimization," *Supplemento ai Rendiconti del Circolo Matematico di Palermo, Serie II,* vol. 58, pp. 21–56, 1999.

[25] V. Mehrmann, *The Autonomous Linear Quadratic Control Problem, Theory and Numerical Solution* (Lecture Notes in Control and Information Sciences, vol. 163). Heidelberg, Germany: Springer-Verlag, 1991.

[26] P.H. Petkov, N.D. Christov, and M.M. Konstantinov, *Computational Methods for Linear Control Systems*. Hertfordshire, U.K.: Prentice-Hall, 1991.

[27] I.M. Jaimoukha and E.M. Kasenally, "Krylov subspace methods for solving large Lyapunov equations," *SIAM J. Numer. Anal.,* vol. 31, no. 1, pp. 227–251, 1994.

[28] P. Benner and H. Faßbender, "An implicitly restarted symplectic Lanczos method for the Hamiltonian eigenvalue problem," *Linear Algebra Appl.*, vol. 263, pp. 75–111, 1997.

[29] T. Penzl, "Numerische Lösung groβer Lyapunov-Gleichungen," dissertation, Logos–Verlag, Berlin, Germany, Fakultät für Mathematik, TU Chemnitz, 1998.

[30] J.-R. Li and J. White, "Low rank solution of Lyapunov equations," *SIAM J. Matrix Anal. Appl.*, vol. 24, no. 1, pp. 260–280, 2002.

[31] A.C. Antoulas, D.C. Sorensen, and Y. Zhou, "On the decay rate of Hankel singular values and related issues," *Syst. Contr. Lett.,* vol. 46, no. 5, pp. 323–342, 2002.

[32] T. Penzl, "Eigenvalue decay bounds for solutions of Lyapunov equations: the symmetric case," *Syst. Control Lett.,* vol. 40, no. 2, pp. 139–144, 2000.

[33] P. Benner, J.-R. Li, and T. Penzl, "Numerical solution of large Lyapunov equations, Riccati equations, and linear-quadratic control problems," Tech. Rep., Fakultät für Mathematik, TU Chemnitz, Germany, 2003.

[34] E.L. Wachspress, "Iterative solution of the Lyapunov matrix equation," *Appl. Math. Lett.,* vol. 107, pp. 87–90, 1988.

[35] S. Gugercin, D.C. Sorensen, and A.C. Antoulas, "A modified low-rank Smith method for large-scale Lyapunov equations," *Numer. Algorithms,* vol. 32, no. 1, pp. 27–55, 2003.

[36] T. Penzl, "LYAPACK Users Guide," Tech. Rep. SFB393/00-33, Sonderforschungsbereich 393 *Numerische Simulation auf massiv parallelen Rechnern,* TU Chemnitz, 09107 Chemnitz, FRG, 2000 [Online]. Available: http://www.tu-chemnitz.de/sfb393/sfb00pr.html

[37] F. Tröltzsch and A. Unger, "Fast solution of optimal control problems in the selective cooling of steel," *Z. Angew. Math. Mech.,* vol. 81, no. 7, pp. 447–456, 2001.

[38] A. Schmidt and K. Siebert, *ALBERT: An Adaptive Hierarchical Finite Element Toolbox*. Institut für Mathematik, University of Freiburg, June 2000.

[39] P. Benner, E.S. Quintana-Ortí, and G. Quintana-Ortí, "Solving linear-quadratic optimal control on parallel computers," , Fakultät für Mathematik, TU Chemnitz, D-09107 Chemnitz (Germany), Tech. Rep., 2004.

[40] J.D. Gardiner and A.J. Laub, "Parallel algorithms for algebraic Riccati equations," *Int. J. Contr.*, vol. 54, no. 6, pp. 1317–1333, 1991.

[41] A.C. Antoulas and A. Astolfi, "H_∞-norm approximation," in *2002 MTNS Problem Book, Open Problems on the Mathematical Theory of Networks and Systems*, V.D. Blondel and A. Megretski, Eds., 2002, pp. 73–76 [Online]. Available: http://www.nd.edu/~mtns/OPMTNS.pdf

[42] A.C. Antoulas, *Lectures on the Approximation of Large-Scale Dynamical Systems.* Philadelphia, PA: SIAM Publications, to be published.

[43] G. Obinata and B.D.O. Anderson, *Model Reduction for Control System Design* (Communications and Control Engineering Series). London, UK: Springer-Verlag, 2001.

[44] A. Varga, "Model reduction software in the SLICOT library," in *Applied and Computational Control, Signals, and Circuits,* B.N. Datta, Ed., Boston, MA: Kluwer, pp. 239–282. 2001.

[45] P. Benner, E.S. Quintana-Ortí, and G. Quintana-Ortí, "Balanced truncation model reduction of large-scale dense systems on parallel computers," *Math. Comput. Model. Dyn. Syst.*, vol. 6, no. 4, pp. 383–405, 2000.

[46] P. Benner, E.S. Quintana-Ortí, and G. Quintana-Ortí, "State-space truncation methods for parallel model reduction of large-scale systems," *Parallel Comput.,* vol. 29, pp. 1701–1722, 2003.

Peter Benner (benner@mathematik.tu-chemnitz.de) received the Diplom in mathematics from RWTH Aachen, Germany, in 1993. From 1993 to 1997 he worked on his Ph.D. at the University of Kansas, Lawrence, and at the TU Chemnitz-Zwickau, Germany, where he received his Ph.D. in 1997. In 2001, he finished his Habilitation at the University of Bremen where he was an assistant professor from 1997 to 2001. He was then a visiting associate professor at the TU Hamburg-Harburg and a lecturer at the TU Berlin. Since 2003, he has been a professor of mathematics in industry and technology at the TU Chemnitz. His research interests are in the areas of scientific computing, numerical mathematics, optimization, and their application in systems and control theory. He can be contacted at Fakultät für Mathematik, Technische Universität Chemnitz, D-09107 Chemnitz, Germany.