Parallel Computation of 3-D Soil-Structure Interaction in Time Domain with a Coupled FEM/SBFEM Approach[*]

Marco Schauer, Sabine Langer

Institut für Angewandte Mechanik, Technische Universität Carolo-Wilhelmina zu Braunschweig, Spielmannstraße 11, 38106 Braunschweig, Germany

Jose E. Roman

D. Sistemes Informàtics i Computació, Universitat Politècnica de València, Camí de Vera s/n, E-46022 València, Spain

Enrique S. Quintana-Ortí

Departamento de Ingeniería y Ciencia de Computadores, Universidad Jaume I, Avda. Sos Baynat s/n, E-12071 Castellón, Spain

Abstract

This paper introduces a parallel algorithm for the scaled boundary finite element method (SBFEM). The application code is designed to run on clusters of computers, and it enables the analysis of large-scale soil-structure-interaction problems, where an unbounded domain has to fulfill the radiation condition for wave propagation to infinity. The main focus of the paper is on the mathematical description and numerical implementation of the SBFEM. In particular, we describe in detail the algorithm to compute the acceleration unit impulse response matrices used in the SBFEM as well as the solvers for the Riccati and Lyapunov equations. Finally, two test cases validate the new code, illustrating the numerical accuracy of the results and the parallel performances.

1 Introduction

Accurate simulations of wave-propagation like aeroacoustic, radar and sonar technology, wireless communication or soil-structure interaction (SSI) through unbounded domains or infinite half-spaces, require careful analysis and call for efficient methods in order to model wave-propagation to infinity. This work focuses on the analysis of SSI. Whenever vibrations or impulses are emitted to soil, they induce waves traveling through the ground that can provoke structures to vibrate and even to fail. Traffic, blasting operations and earthquakes are some of the reasons that generate those kinds of emissions. Basically there are two major motivations: in active seismic areas reliable earthquake resistant structures are required, and in our urban society a very important and challenging task is to encapsulate a building from the surrounding emissions to increase its comfort.

[*]Jose E. Roman and Enrique S. Quintana-Ortí were partially supported by the Spanish Ministerio de Ciencia e Innovación under grants TIN2009-07519, and TIN2008-06570-C04-01, respectively.

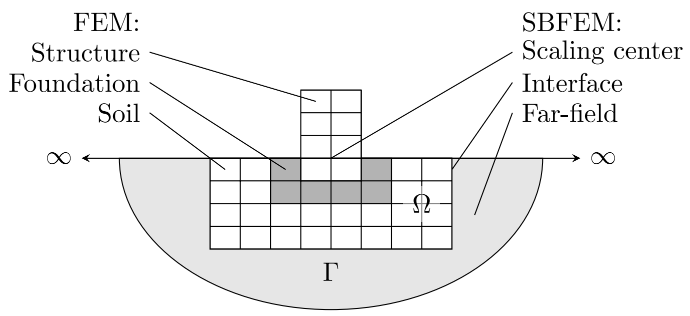


Figure 1: Problem definition

In both cases it is essential not only to analyze the structure itself but also to take the surrounding soil into account [2, 25, 28].

Analyzing SSI questions is complex since two very different mechanical problems have to be solved: the structure itself and the infinite half-space, which surrounds the domain of interest. Usually the engineers have to deal with complex geometries or materials as well. For those complex problem types no analytical or even semi-analytical solution is available, which asks for the use of computational models. The structure can be discretized by using standard methods like finite differences or finite elements. In contrast, the infinite half-space cannot be analyzed with those standard methods as they do not fulfill the radiation condition. Because the standard methods are neither able to discretize infinite domains nor to satisfy the radiation condition, it seems natural to divide the problem into two separate subproblems.

By using a substructuring method (see Figure 1), the soil-structure system is divided into two subdomains. The structure and its foundation as well as parts of the surrounding soil represent the near-field, denoted by Ω. The unbounded soil around the near-field can be represented by a far-field for which the radiation condition has to be fulfilled. Both fields, near and far, are coupled at the interface Γ. The near-field can be easily represented by finite elements, and the far-field by boundary element method (BEM) or a scaled boundary finite element method (SBFEM) [32, 31], which yields to a direct coupling of structure and the surrounding unbounded domain. The SBFEM is used in the following because it combines the advantages of both FEM and BEM [22]. Like the FEM, the SBFEM does not require a fundamental solution and the coefficient matrices are symmetric and can be added to the FEM matrices without changing their size. On the other hand, at the boundary the spatial dimension is reduced by one and the radiation conditions are satisfied exactly, as they are in the BEM [31]. Another major advantage is that the coefficient matrices can be reused, once they are computed. This allows the user to modify the setup of the near-field, e.g., buildings, infrastructure and loadings, without computing the coefficient matrices again. The only limitation is that the material parameters at the interface Γ must not change. The SBFEM is also applicable to non-linear analysis. In this case, the coefficient matrices have to be computed during the coupled FEM-SBFEM computation.

Other methods that consider the surrounding unbounded domain by a transmitting boundary, which approximates the radiation condition, have been developed during the last years. The viscous boundary condition is one of the simplest transmitting boundary conditions, acting like a dashpot [24]. Other local, arbitrary order absorbing boundary conditions [15, 23, 3] and several other types of transmitting boundaries (e.g., infinite elements [11, 4]) have been proposed, but none of them is able to fulfill the radiation condition exactly.

When a basic SSI problem consisting of a geometrically simple foundation and the surrounding space is discretized, the number of degrees of freedom is small. Such problems can generally be analyzed using a standard desktop PC. As the size of the foundation, the structure itself, and their complexity grow, the necessary number of degrees of freedom increases as well to yield the same accuracy in the results. Handling a large number of degrees of freedom on a standard PC results in over-proportionally increased computation time. More often, this is even impossible due to the lack of sufficient memory. Hence, the options for analyzing large SSI problems are either to simplify the model, at the expense of loss in accuracy, or to keep the detailed model and use parallel computing techniques.

Publications concerning the application of SBFEM to large scale or larger scaled problems in time-domain, where the far-field is discretized by only one sub-domain, are not known to the authors. SSI analysis using substructured far-fields have been discussed in the literature [31, 13, 30]. This work presents a way to realize a coupled FEM-SBFEM approach for large scale problems without sub-structuring the interface Γ, so that no artificial boundaries have to be introduced which would cause invalid physical behavior.

Parallel computing combines several computational cores, which can potentially lead to a significant reduction of computation time, enabling the analysis of complex geometries. As the costs for so-called PC clusters have dropped considerably in the past years while computing capabilities have been improved constantly, parallel computation has become more popular and is turning into a state-of-the-art tool to handle large-scale problems. Developing efficient parallel codes for matrix computations can be difficult, but much of this programming burden is removed thanks to existing numerical libraries.

The paper is structured as follows. In section 2 the FEM/SBFEM coupling is introduced briefly and a more detailed description of the SBFEM part follows in section 2.1. The algorithm to solve the SBFEM part and its implementation for time domain simulations is discussed in detail in section 3. This section also includes a summary of the libraries we use to implement the software. Numerical tests to validate the code and to illustrate its parallel performance are presented in sections 4 and 5, respectively. We wrap up with a brief discussion in section 6.

2 Coupled FEM/SBFEM approach

In the following, we shortly introduce the FEM and SBFEM used to model the near-field and far-field, respectively. The displacement-based finite element method at an arbitrary time step can be written as

$$\mathbf{M}\frac{d^2\mathbf{u}}{dt^2} + \mathbf{C}\frac{d\mathbf{u}}{dt} + \mathbf{K}\mathbf{u} = \mathbf{p}, \tag{1}$$

where the vector $\mathbf{u}$ represents the nodal displacement, $\frac{d\mathbf{u}}{dt} = \dot{\mathbf{u}}$ nodal velocity, $\frac{d^2\mathbf{u}}{dt^2} = \ddot{\mathbf{u}}$ stands for the nodal acceleration, and $\mathbf{p}$ denotes the applied nodal forces. Here, $\mathbf{M}$ is the mass matrix, $\mathbf{C}$ is the damping matrix and $\mathbf{K}$ denotes the stiffness matrix. The damping matrix is realized as Rayleigh damping, hence $\mathbf{C} = c_m\mathbf{M} + c_k\mathbf{K}$. Consider the time period T divided into n time steps with the duration $\Delta t = \frac{T}{n}$; the application of the implicit time integration scheme Hilbert-Hughes-Taylor-α (HHT-α) then yields [19]

$$\mathbf{M}\ddot{\mathbf{u}}_{t_{n+1}} + (1+\alpha)\mathbf{C}\dot{\mathbf{u}}_{t_{n+1}} - \alpha\mathbf{C}\dot{\mathbf{u}}_{t_n} + (1+\alpha)\mathbf{K}\mathbf{u}_{t_{n+1}} - \alpha\mathbf{K}\mathbf{u}_{t_n} = \mathbf{p}_{t_{n+1}+\alpha\Delta t}, \tag{2}$$

$$\mathbf{u}_{t_{n+1}} = \mathbf{u}_{t_n} + \Delta t \dot{\mathbf{u}}_{t_n} + \Delta t^2 \frac{1}{2} - \beta \ddot{\mathbf{u}}_{t_n} + \beta \Delta t^2 \ddot{\mathbf{u}}_{t_{n+1}}, \tag{3}$$

$$\dot{\mathbf{u}}_{t_{n+1}} = \dot{\mathbf{u}}_{t_n} + (1-\gamma)\Delta t \ddot{\mathbf{u}}_{t_n} + \gamma \Delta t \ddot{\mathbf{u}}_{t_{n+1}}. \tag{4}$$

Thus, for $\alpha = 0$, the HHT-α scheme equals the Newmark integration method [27].

To couple the FEM with the SBFEM, the entries of the matrices in (1) have to be split into the near- and far-fields, which results in the FEM equation:

$$\begin{bmatrix} \mathbf{M}_{nn} & \mathbf{M}_{nf} \\ \mathbf{M}_{fn} & \mathbf{M}_{ff} \end{bmatrix} \ddot{\mathbf{u}} + \begin{bmatrix} \mathbf{C}_{nn} & \mathbf{C}_{nf} \\ \mathbf{C}_{fn} & \mathbf{C}_{ff} \end{bmatrix} \dot{\mathbf{u}} + \begin{bmatrix} \mathbf{K}_{nn} & \mathbf{K}_{nf} \\ \mathbf{K}_{fn} & \mathbf{K}_{ff} \end{bmatrix} \mathbf{u} = \begin{bmatrix} \mathbf{p}_{nn} \\ \mathbf{p}_{ff} \end{bmatrix} - \begin{bmatrix} \mathbf{0} \\ \mathbf{p}_b \end{bmatrix}. \tag{5}$$

In this expression, blocks with subscript "nn" contain nodes at the near-field while blocks with subscript "ff" comprise nodes at the far-field. The coupling of near-field nodes and far-field nodes is reflected in those blocks marked with the subscripts "nf" and "fn". Vector $\mathbf{p}_b$, which acts only at the boundary Γ (see Figure 1), denotes the far-field influence on the near-field, so that the behaviour of the infinite half space can be applied to the FEM subdomain as a load.

The second part of this sub-structuring approach is the far-field represented by the SBFEM. The forces $\mathbf{p}_b$ at the near-field/far-field interface are given by the convolution integral

$$\mathbf{p}_b(t) = \int_0^t \mathbf{M}^\infty(t-\tau)\ddot{\mathbf{u}}(\tau)d\tau, \tag{6}$$

where $\mathbf{M}^\infty(t)$ is the acceleration unit-impulse response matrix, also known as the influence matrix. In order to solve the convolution integral equation (6) in time domain, a piecewise constant acceleration unit impulse response matrix is assumed, i.e.,

$$\mathbf{M}^\infty(t) = \begin{cases} \mathbf{M}_0^\infty & t \in [0; \Delta t], \\ \mathbf{M}_1^\infty & t \in [\Delta t; 2\Delta t], \\ \vdots & \vdots \\ \mathbf{M}_n^\infty & t \in [(n-1)\Delta t; n\Delta t], \end{cases} \tag{7}$$

so that equation (6) can be rewritten in discrete form as

$$\mathbf{p}_b(t_n) = \sum_{j=1}^{n} \mathbf{M}_{n-j}^\infty \int_{(j-1)\Delta t}^{j\Delta t} \ddot{\mathbf{u}}(\tau)d(\tau). \tag{8}$$

This equation is then transformed using HHT-α (with γ parameter of HHT-α scheme) into

$$\mathbf{p}_b(t_n) = \gamma \Delta t \mathbf{M}_0^\infty \ddot{\mathbf{u}}_n + \sum_{j=1}^{n-1} \mathbf{M}_{n-j}^\infty (\dot{\mathbf{u}}_j - \dot{\mathbf{u}}_{j-1}). \tag{9}$$

The coupling of FEM and SBFEM is done at the common interface of both subdomains with the SBFEM part (9) being simply added to the sorted FEM part (5):

$$\begin{bmatrix} \mathbf{M}_{nn} & \mathbf{M}_{nf} \\ \mathbf{M}_{fn} & \mathbf{M}_{ff} + \gamma \Delta t \mathbf{M}_0^\infty \end{bmatrix} \ddot{\mathbf{u}} + \begin{bmatrix} \mathbf{C}_{nn} & \mathbf{C}_{nf} \\ \mathbf{C}_{fn} & \mathbf{C}_{ff} \end{bmatrix} \dot{\mathbf{u}} + \\ \begin{bmatrix} \mathbf{K}_{nn} & \mathbf{K}_{nf} \\ \mathbf{K}_{fn} & \mathbf{K}_{ff} \end{bmatrix} \mathbf{u} = \begin{bmatrix} \mathbf{p}_{nn} \\ \mathbf{p}_{ff} - \sum\limits_{j=1}^{n-1} \mathbf{M}_{n-j}^\infty (\dot{\mathbf{u}}_j - \dot{\mathbf{u}}_{j-1}) \end{bmatrix}. \tag{10}$$

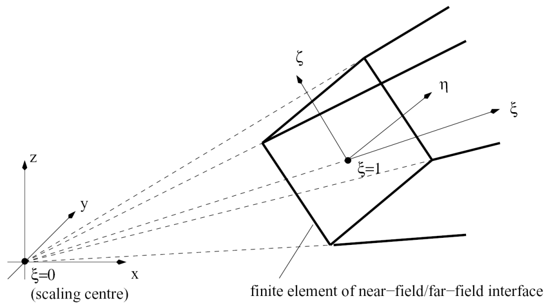


Figure 2: Scaled boundary transformation for three-dimensional problems: finite element and coordinate systems.

The discretization is now fully described. The following section is concerned with the formulation of the SBFEM in time domain and the computation of the acceleration unit-impulse response matrices $\mathbf{M}^{\infty}(t)$ which is needed in equation (10).

2.1 SBFEM formulation

The scaled boundary finite elements are described by a local coordinate system, η and ζ, on the boundary and the radial coordinate ξ; see Figure 2. This system (ξ, η, ζ) is linked with the Cartesian coordinate system $\mathbf{x}_i$; in particular, ξ represents the distance between the scaling center $(\xi = 0)$, boundary $(\xi = 1)$ and infinity.

A finite elements geometry at the boundary is represented by interpolating its nodal coordinates $\mathbf{x}_i$ with the shape functions $N(\eta, \zeta)$ using the local coordinates η and ζ. An arbitrary point $\mathbf{x}(\xi, \eta, \zeta)$ can be described by scaling its corresponding point at the boundary with the radial coordinate ξ. As the displacements, stresses and strains are defined in Cartesian coordinates, the differential operator $\mathbf{D}$ has to be modified, so that the governing equation of elastodynamics is transformed to the scaled boundary coordinate system. For details, see [32, 31, 21].

2.1.1 Frequency domain

Although the SBFEM is used in time domain, its derivation initiates in frequency domain. Therefore, in the following we will briefly describe the latter and postpone the presentation in time domain to subsection 2.1.2. For full details on the computational approach to the three dimensional wave propagation, see, e.g., [32].

The dynamic equilibrium in frequency domain reads

$$\mathbf{D}^T \hat{\sigma} + \hat{\mathbf{f}} + \omega^2 \rho \hat{\mathbf{u}} = 0, \tag{11}$$

where $\mathbf{D}$ is the operator matrix, and vectors $\hat{\sigma}$, $\hat{\mathbf{f}}$ and $\hat{\mathbf{u}}$ represent the amplitudes of stress, body force and displacement, respectively; the frequency is given by ω and ρ denotes the material density.

The FEM is formulated in a nodal force-displacement relationship. Therefore, the SBFEM has to be formulated in the same way to obtain an analog expression. At all surfaces Γ_ξ the displacement $\mathbf{u}(\xi)$ can be determined by applying the shape function $N(\eta, \zeta)$. The mapping equation

$$\mathbf{u}(\xi, \eta\zeta) = N(\eta, \zeta)\mathbf{u}(\xi) \tag{12}$$

allows to derive strains

$$\epsilon(\xi,\eta,\zeta) = \mathbf{B}_1\mathbf{u}(\xi)_{,\xi} + \frac{1}{\xi}\mathbf{B}_2\mathbf{u} \tag{13}$$

as well as stresses

$$\sigma(\xi,\eta,\zeta) = \mathbf{E}\epsilon(\xi,\eta,\zeta). \tag{14}$$

Here, $\mathbf{E}$ denotes the elasticity matrix, and $\mathbf{B}_1$, $\mathbf{B}_2$ are operator matrices that contain the evaluated shape functions and their derivations. Introducing equations (12), (13) and (14) in (11), performing the integration by parts, and introducing the element coefficient matrices

$$\mathbf{C}_1 = \int_{-1}^{+1}\int_{-1}^{+1} \mathbf{B}_1^T\mathbf{E}\mathbf{B}_1|\mathbf{J}|d\eta d\zeta, \tag{15}$$

$$\mathbf{C}_2 = \int_{-1}^{+1}\int_{-1}^{+1} \mathbf{B}_2^T\mathbf{E}\mathbf{B}_1|\mathbf{J}|d\eta d\zeta, \tag{16}$$

$$\mathbf{C}_3 = \int_{-1}^{+1}\int_{-1}^{+1} \mathbf{B}_2^T\mathbf{E}\mathbf{B}_2|\mathbf{J}|d\eta d\zeta, \quad \text{and} \tag{17}$$

$$\mathbf{M} = \int_{-1}^{+1}\int_{-1}^{+1} \rho\mathbf{N}^T\mathbf{N}|\mathbf{J}|d\eta d\zeta, \tag{18}$$

the scaled boundary finite element equation in displacement formulation is given by

$$\begin{gathered}\mathbf{C}_1\xi^2\mathbf{u}_{,\xi\xi}(\xi) + \left(2\mathbf{C}_1 - \mathbf{C}_2 + \mathbf{C}_2^T\right)\xi\mathbf{u}_{,\xi}(\xi) + \left(\mathbf{C}_2^2 - \mathbf{C}_3\right)\mathbf{u}(\xi) + \\ \omega^2\mathbf{M}\xi^2\mathbf{u}(\xi) + \xi^2\mathbf{f}_b(\xi) = \mathbf{0}\end{gathered} \tag{19}$$

This is the weak form of differential equation of motion and it is valid for bounded ($0 \leq \xi \leq 1$) and unbounded ($1 \leq \xi \leq \infty$) domains as well. The next step is to formulate this equation using the dynamic stiffness matrix $\mathbf{S}^\infty$ for the unbounded medium at the boundary ($\xi = 1$), that yields

$$\left(\mathbf{S}^\infty(\omega) + \mathbf{C}_2\right)\mathbf{C}_1^{-1}\left(\mathbf{S}^\infty(\omega) + \mathbf{C}_2^T\right) - \mathbf{S}^\infty(\omega) - \omega\mathbf{S}^\infty(\omega)_{,\omega} - \mathbf{C}_3 + \omega^2\mathbf{M} = \mathbf{0}. \tag{20}$$

This is a non-linear first order differential equation, which represents the scaled boundary finite element equation in frequency domain for three dimensional elastodynamics. $\mathbf{S}^\infty(\omega)$ is the unknown dynamic stiffness matrix for the unbounded medium, with the independent frequency ω [32].

2.1.2 Time domain

In time domain analysis the acceleration unit impulse response matrix $\mathbf{M}^\infty(t)$ is required and equation (20) has to be transformed into this domain. The Fourier transformation directly relates $\mathbf{M}^\infty(t)$ and $\mathbf{S}^\infty(\omega)$, which is given as

$$\mathbf{M}^\infty(\omega) = \frac{\mathbf{S}^\infty(\omega)}{(i\omega)^2}. \tag{21}$$

Inserting $\mathbf{M}^{\infty}(\omega)$ and applying the Fourier transformation to equation (20) leads to

$$\begin{gathered}\int_0^t \tilde{\mathbf{M}}^{\infty}(t-\tau)\tilde{\mathbf{M}}^{\infty}(\tau)d\tau + \tilde{\mathbf{C}}_2 \int_0^t\int_0^\tau \tilde{\mathbf{M}}^{\infty}(\tau')d\tau' d\tau + \int_0^t\int_0^\tau \tilde{\mathbf{M}}^{\infty}(\tau')d\tau' d\tau \tilde{\mathbf{C}}_2^T + \\ t\int_0^t \tilde{\mathbf{M}}^{\infty}(\tau)d\tau - \tfrac{t^3}{6}\tilde{\mathbf{C}}_3\mathbf{H}(t) - t\tilde{\mathbf{M}}\mathbf{H}(t) = \mathbf{0}\end{gathered} \tag{22}$$

with coefficient matrices

$$\begin{aligned}\mathbf{C}_1 &= \mathbf{L}\mathbf{L}^T, && (23)\\ \tilde{\mathbf{C}}_2 &= \mathbf{L}^{-1}\mathbf{C}_2\mathbf{L}^{-T}, && (24)\\ \tilde{\mathbf{C}}_3 &= \mathbf{L}^{-1}\left(\mathbf{C}_3 - \mathbf{C}_2\mathbf{C}_1^{-1}\mathbf{C}_2^T\right)\mathbf{L}^{-T}, && (25)\\ \tilde{\mathbf{M}} &= \mathbf{L}^{-1}\mathbf{M}\mathbf{L}^{-T}, && (26)\end{aligned}$$

and the Heaviside step function $\mathbf{H}(t)$. Thus, $\mathbf{L}$ results from a Cholesky decomposition of $\mathbf{C}_1$; the time dependent matrix $\tilde{\mathbf{M}}^{\infty}(t)$ can be approximated for each time step by equation (22); and the acceleration unit impulse response matrix $\mathbf{M}^{\infty}(t)$ can be derived from

$$\mathbf{M}^{\infty}(t) = \mathbf{L}\tilde{\mathbf{M}}^{\infty}(t)\mathbf{L}^T. \tag{27}$$

First time step. Solving the convolution integral of equation (22) leads to a quadratic equation in the unknown matrix $\tilde{\mathbf{M}}^{\infty}$ at the first time step:

$$\tilde{\mathbf{M}}_0^{\infty 2} + \frac{\Delta t}{2}\left(\tilde{\mathbf{C}}_2 + \mathbf{I}\right)\tilde{\mathbf{M}}_0^{\infty} + \tilde{\mathbf{M}}_0^{\infty}\frac{\Delta t}{2}\left(\tilde{\mathbf{C}}_2^T + \mathbf{I}\right) - \frac{\Delta t^2}{6}\tilde{\mathbf{C}}_3 - \tilde{\mathbf{M}} = \mathbf{0} \tag{28}$$

This is analogous to an algebraic Riccati equation of the form

$$\mathbf{F}\mathbf{X} + \mathbf{X}\mathbf{F}^T - \mathbf{X}\mathbf{B}\mathbf{B}^T\mathbf{X} + \mathbf{Q} = \mathbf{0}, \tag{29}$$

where $\mathbf{F} = -\frac{\Delta t}{2}\left(\tilde{\mathbf{C}}_2 + \mathbf{I}\right)$, $\mathbf{B} = \mathbf{I}$ the identity matrix, and $\mathbf{Q} = \frac{\Delta t^2}{6}\tilde{\mathbf{C}}_3 + \tilde{\mathbf{M}}$. The solution is $\mathbf{X} = \tilde{\mathbf{M}}_0^{\infty}$. This quadratic equation appears in the first time step only, so this time step requires a specialized procedure.

n-th time step. All subsequent time steps, t_n with $n \geq 1$, are given by

$$\begin{gathered}\left(\tilde{\mathbf{M}}_0^{\infty} + \tfrac{\Delta t}{2}\tilde{\mathbf{C}}_2\right)\tilde{\mathbf{M}}_n^{\infty} + \tilde{\mathbf{M}}_n^{\infty}\left(\tilde{\mathbf{M}}_0^{\infty} + \tfrac{\Delta t}{2}\tilde{\mathbf{C}}_2^T\right) + t\tilde{\mathbf{M}}_n^{\infty} = \\ -\sum_{k=1}^{n-1}\left(\tilde{\mathbf{M}}_{n-k}^{\infty}\tilde{\mathbf{M}}_k^{\infty}\right) - \tilde{\mathbf{C}}_2\left(\tfrac{\mathbf{J}_{n-1}}{\Delta t} + \mathbf{I}_{n-1}\right) \\ -\left(\tfrac{\mathbf{J}_{n-1}}{\Delta t} + \mathbf{I}_{n-1}\right)\tilde{\mathbf{C}}_2^T + \tfrac{t^3}{6\Delta t}\tilde{\mathbf{C}}_3 + \tfrac{t}{\Delta t}\left(\tilde{\mathbf{M}} - \mathbf{I}_{n-1}\right)\end{gathered} \tag{30}$$

considering the integrals

$$\mathbf{I}_n = \int_0^{n\Delta t}\tilde{\mathbf{M}}^{\infty}(\tau)d\tau = \mathbf{I}_{n-1} + \Delta t\tilde{\mathbf{M}}_n^{\infty} \tag{31}$$

and

$$\mathbf{J}_n = \int_0^{n\Delta t} \int_0^{\tau} \tilde{\mathbf{M}}^{\infty}(\tau')d\tau' d\tau = \mathbf{J}_{n-1} + \Delta t \mathbf{I}_{n-1} + \frac{\Delta t^2}{2} \tilde{\mathbf{M}}_n^{\infty}. \tag{32}$$

In this case, (30) is a Lyapunov equation of the form

$$\tilde{\mathbf{A}}\mathbf{X} + \mathbf{X}\tilde{\mathbf{A}}^T = \mathbf{C}, \tag{33}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \frac{t}{2}\mathbf{I}$ with $\mathbf{A} = \tilde{\mathbf{M}}_0^{\infty} + \frac{\Delta t}{2}\tilde{\mathbf{C}}_2$, and $\mathbf{C}$ is equal to the expression on the right hand side of (30). The solution is $\mathbf{X} = \tilde{\mathbf{M}}_n^{\infty}$.

3 Algorithm and its implementation

In this section, we survey practical aspects related to the methods and software tools required to implement them. Specifically, we concentrate on the algorithm to compute the acceleration unit impulse response matrices $\mathbf{M}_n^{\infty}$ and the solvers that are employed for the Riccati and Lyapunov matrix equations. We assume that the size of relevant problems ranges from moderate to large, thus calling for parallel computing capabilities.

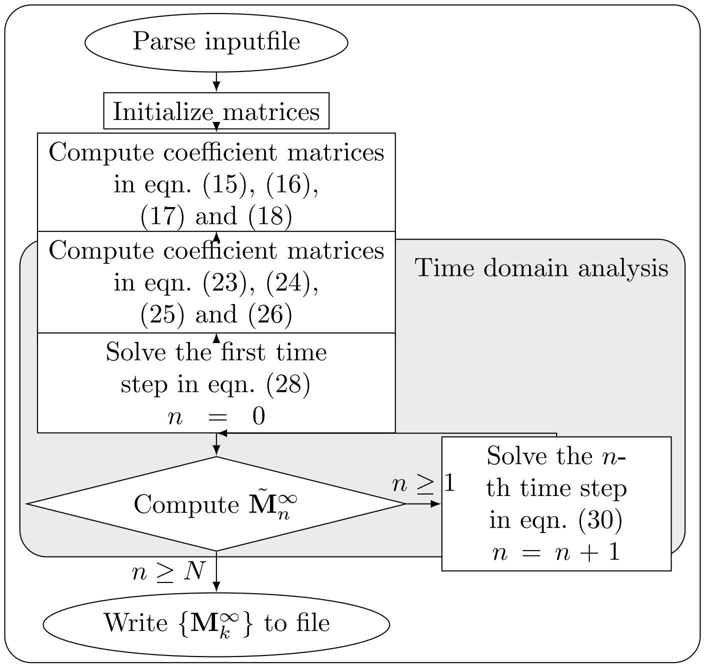
3.1 Global algorithm and main building blocks

The flowchart in Figure 3 summarizes the operations required to compute the $\mathbf{M}_n^{\infty}$ matrices. The first consideration when attempting to implement this algorithm is whether the sparsity of matrices can be exploited. The short answer is no. The matrices resulting from the SBFEM, (15)–(18), are sparse and must be assembled and stored in some sparse format. However, sparsity is lost when the coefficient matrices of the time domain formulation, (23)–(26), are computed. One could think of maintaining matrices such as $\tilde{\mathbf{C}}_2$ in implicit form, with a sparse Cholesky factor $\mathbf{L}$. However, even if this was feasible, it is not possible to compute low-rank approximations to the solution of the matrix equations (Riccati and Lyapunov). Consequently, most of the algorithm must be addressed with techniques pertaining to dense linear algebra. Fortunately, the resulting matrices $\mathbf{M}_n^{\infty}$ are again sparse, enabling their conversion to sparse format. This is crucial for reducing the complexity of subsequent computations related to the coupling of near-field and far-field, as described in Section 2.

The consequence is that we need to develop a hybrid sparse-dense code that operates in parallel. This adds a lot of complexity to programming since it requires the integration of numerical libraries of different nature, and the use of the appropriate data structures in each case.

We now detail the main steps of the computation in algorithmic form. The complete time domain analysis is shown in Algorithm 1. From the computational viewpoint, the most expensive operations are the computation of the Schur decomposition, in [1.9], and the solution of the matrix equations associated with the computation of the first acceleration unit impulse response matrix $\tilde{\mathbf{M}}_0^{\infty}$, in [1.5], and subsequent time steps, in [1.11]. The matrix equations solvers will be described in detail in subsections 3.2, 3.3 and 3.4. Before doing so, we discuss some other important implementation issues.

The storage requirements of the algorithm are high, because the number of matrices is considerable. As mentioned before, the input matrices $\mathbf{C}_1$, $\mathbf{C}_2$, $\mathbf{C}_3$ and $\mathbf{M}$ are sparse. In [1.1]–[1.4] we shift to dense storage. These operations correspond to equations (23)–(26) (the expression for $\tilde{\mathbf{C}}_3$ has been simplified). Sparse storage is recovered only in [1.13]. All $\tilde{\mathbf{M}}_n^{\infty}$ need to be kept for later



Parse inputfile
Initialize matrices
Compute coefficient matrices in eqn. (15), (16), (17) and (18)
Time domain analysis
Compute coefficient matrices in eqn. (23), (24), (25) and (26)
Solve the first time step in eqn. (28) $n = 0$
Compute $\tilde{\mathbf{M}}_n^\infty$
$n \geq 1$
Solve the n-th time step in eqn. (30) $n = n + 1$
$n \geq N$
Write $\{\mathbf{M}_k^\infty\}$ to file

Figure 3: Flowchart to compute the acceleration unit impulse response matrix $\mathbf{M}^\infty$.

Algorithm 1 (Time domain analysis)

Input: timestep Δt, maximum number of iterations N, coefficient matrices $\mathbf{C}_1$, $\mathbf{C}_2$, $\mathbf{C}_3$, $\mathbf{M}$
Output: $\{\mathbf{M}_n^\infty\}_{n=0,\dots,N}$

```
[1.1]  Compute the Cholesky factorization $\mathbf{C}_1 = \mathbf{L}\mathbf{L}^T$
[1.2]  $\tilde{\mathbf{C}}_2 = \mathbf{L}^{-1}\mathbf{C}_2\mathbf{L}^{-T}$
[1.3]  $\tilde{\mathbf{C}}_3 = \mathbf{L}^{-1}\mathbf{C}_3\mathbf{L}^{-T} - \tilde{\mathbf{C}}_2\tilde{\mathbf{C}}_2^T$
[1.4]  $\tilde{\mathbf{M}} = \mathbf{L}^{-1}\mathbf{M}\mathbf{L}^{-T}$
[1.5]  Perform 1st time step (Algorithm 2)
[1.6]  $\mathbf{I}_0 = \Delta t\tilde{\mathbf{M}}_0^\infty$
[1.7]  $\mathbf{J}_0 = \frac{\Delta t2}{2}\tilde{\mathbf{M}}_0^\infty$
[1.8]  $\mathbf{A} = \tilde{\mathbf{M}}_0^\infty + \frac{\Delta t}{2}\tilde{\mathbf{C}}_2$
[1.9]  Compute the Schur decomposition $\mathbf{AV} = \mathbf{VS}$
[1.10] for $n = 1, 2, \dots, N$
[1.11]     Perform $n$-th time step (Algorithm 5) for $t = (n+1)\Delta t$
[1.12] end for
[1.13] For each computed matrix, $\mathbf{M}_n^\infty = \text{sparse}(\mathbf{L}\tilde{\mathbf{M}}_n^\infty\mathbf{L}^T)$
```

computations. In contrast, matrices $\mathbf{I}_n$ and $\mathbf{J}_n$ can be computed in-place (overwriting the same memory locations) since they depend only on the previous time step.

Algorithm 2 (First time step)

```
Input: timestep $\Delta t$, coefficient matrices $\tilde{\mathbf{C}}_2$, $\tilde{\mathbf{C}}_3$, $\tilde{\mathbf{M}}$
Output: $\tilde{\mathbf{M}}_0^\infty$
  [2.1] $\mathbf{F} = -\frac{\Delta t}{2}\left(\tilde{\mathbf{C}}_2^T + \mathbf{I}\right)$
  [2.2] $\mathbf{B} = I$
  [2.3] $\mathbf{Q} = \frac{\Delta t^2}{6}\tilde{\mathbf{C}}_3 + \tilde{\mathbf{M}}$
  [2.4] Solve the Riccati equation (29) for $\tilde{\mathbf{M}}_0^\infty$ (Algorithm 3)
```

3.2 Solution of Riccati matrix equations

In [20], Kleinmann shows that Newton's root-finding iteration, applied to the (algebraic) Riccati equation, converges under mild conditions to the desired stabilizing solution. In particular, let

$$\mathcal{R}(\mathbf{X}^*) = \mathbf{F}\mathbf{X}^* + \mathbf{X}^*\mathbf{F}^T - \mathbf{X}^*\mathbf{B}\mathbf{B}^T\mathbf{X}^* + \mathbf{Q}. \tag{34}$$

Then, Newton's method for the Riccati equation can be formulated as shown in Algorithm 3. The basic idea is to improve the current approximate solution $\mathbf{X}_k$ by a correction $\tilde{\mathbf{X}}$ obtained by solving the Lyapunov equation

$$\mathbf{A}_k\tilde{\mathbf{X}} + \tilde{\mathbf{X}}\mathbf{A}_k^T + \mathcal{R}(\mathbf{X}_k) = 0, \tag{35}$$

with $\mathbf{A}_k = \mathbf{F} - \mathbf{B}\mathbf{B}^T\mathbf{X}_k$.

Algorithm 3 (Newton method for the Riccati equation)

```
Input: coefficient matrices $\mathbf{F}$, $\mathbf{B}$, $\mathbf{Q}$, and a user defined tolerance $\tau_r$
Output: approximated solution $\mathbf{X} = \mathbf{X}_k$ of the Riccati equation (29) after $k$ Newton steps
  [3.1] Choose some initial starting guess $\mathbf{X}_0 = \mathbf{X}_0^T$
  [3.2] $k = 0$
  [3.3] while $\|\mathcal{R}(\mathbf{X}_k)\|_F > \tau_r \cdot \|\mathbf{X}_k\|_F$
  [3.4]     $\mathbf{A}_k = \mathbf{F} - \mathbf{B}\mathbf{B}^T\mathbf{X}_k$
  [3.5]     Solve for $\tilde{\mathbf{X}}$ the Lyapunov equation $\mathbf{A}_k\tilde{\mathbf{X}} + \tilde{\mathbf{X}}\mathbf{A}_k^T + \mathcal{R}(\mathbf{X}_k) = 0$
  [3.6]     $\mathbf{X}_{k+1} = \mathbf{X}_k + \tilde{\mathbf{X}}$
  [3.7]     $k = k + 1$
  [3.8] end while
```

The rate of convergence of Newton's method strongly depends on the distance between the initial guess $\mathbf{X}_0$ and the exact solution. In our case, as $\mathbf{F}$ is stable (i.e., all its eigenvalues have negative real part), $\mathbf{X}_0 = \|\mathbf{F}\|_1 \cdot I$, with I the identity matrix, is a feasible initial guess and Newton's method (with line search) becomes a competitive alternative as a solver for the algebraic Riccati equation; see [7, 8].

In our implementations we set the iteration tolerance threshold $\tau_r = c_r \cdot \sqrt{\varepsilon}$, where c_r is a constant depending mildly on n, and ε is the machine precision. Once the stopping criterion is satisfied, we perform two additional iterations. Due to the quadratic convergence of Newton's method this is usually enough to reach the attainable accuracy and avoids possible stagnation of the method [6, 10].

Newton's method for the Riccati equation is composed of highly-parallel matrix-matrix products, as in the computation of $\mathcal{R}(\mathbf{X}_k)$ in [3.3] and the coefficient matrix $\mathbf{A}_k$ in [3.4]. The major computation, however, is the solution of the Lyapunov equation in [3.5]. Fortunately, $\mathbf{A}_k$ is stable and, therefore, this equation can be efficiently solved via the matrix-sign function on a parallel architecture, as explained in the next subsection.

3.3 Solution of Lyapunov matrix equations via the matrix sign function

Roberts [29] was the first to use the matrix sign function for solving Lyapunov equations with stable coefficient matrix $\mathbf{A}$. In particular, Roberts shows that, when applied to the Newton matrix sign function, this yields the procedure in Algorithm 4 for the solution of the Lyapunov equation.

Algorithm 4 (Newton method for the Lyapunov equation)

Input: coefficient matrix $\mathbf{A}$ (stable), right hand side $\mathbf{Q} = -\mathcal{R}(\mathbf{X}_k)$, user-defined tolerance τ_l
Output: approximated solution $\tilde{\mathbf{X}}$ of the Lyapunov equation (35) after k Newton steps

```
[4.1] $\mathbf{A}_0 = \mathbf{A}$
[4.2] $\mathbf{Q}_0 = \mathbf{Q}$
[4.3] $k = 0$
[4.4] repeat
[4.5]     $\mathbf{A}_{k+1} = \frac{1}{2}\left(\mathbf{A}_k + \mathbf{A}_k^{-1}\right)$
[4.6]     $\mathbf{Q}_{k+1} = \frac{1}{2}\left(\mathbf{Q}_k + \mathbf{A}_k^{-1}\mathbf{Q}_k\mathbf{A}_k^{-T}\right)$
[4.7]     $k = k + 1$
[4.8] until $\|\mathbf{A}_k - \mathbf{A}_{k-1}\|_F \; > \; \tau_l \cdot \|\mathbf{A}_k\|_F$
[4.9] $\tilde{\mathbf{X}} = \frac{1}{2}\mathbf{Q}$
```

As in the Newton iterative scheme for the Riccati equation, in our implementations we set the tolerance $\tau_l = c_l \cdot \sqrt{\varepsilon}$, where c_l depends mildly on n. Once the stopping criterion is satisfied, two additional iterations are performed to ensure that the attainable accuracy is reached.

Algorithm 4 basically consists of matrix inversion and matrix-matrix products which can be efficiently implemented in current parallel platforms.

3.4 Solution of Lyapunov matrix equations via the Schur method

In terms of dense computations, we can derive a quasi-triangular form of the Lyapunov equation (33) by using the (real) Schur form. In this particular case, this transformation represents a huge computational saving. (Note that the same does not hold for the Lyapunov equation that has to be solved at each step of Newton iteration for the Riccati equation as, there, the coefficient matrix $\mathbf{A}_k$ changes at every iteration.) We can write equation (33) as

$$\tilde{\mathbf{A}}\mathbf{X} + \mathbf{X}\tilde{\mathbf{A}}^T = \mathbf{C} \qquad (36)$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \frac{1}{2}tI$. Recall that the real Schur decomposition of a matrix $\mathbf{A}$ is defined as $\mathbf{AV} = \mathbf{VS}$, where $\mathbf{V}$ is an orthogonal matrix, $\mathbf{V}^T\mathbf{V} = I$, and $\mathbf{S}$ is an upper quasi-triangular matrix with 1×1 or 2×2 diagonal blocks containing real eigenvalues or complex conjugate eigenvalues of $\mathbf{A}$, respectively. The Schur decomposition is invariant under shifts of origin, that is, the Schur form of $\mathbf{A}+\alpha I$ is $\mathbf{S}+\alpha I$. This allows us to reuse the computationally expensive Schur form of $\mathbf{A}$ in all subsequent time steps. The resulting Lyapunov equation is

$$\tilde{\mathbf{S}}\tilde{\mathbf{X}}+\tilde{\mathbf{X}}\tilde{\mathbf{S}}^T=\tilde{\mathbf{C}} \tag{37}$$

where $\tilde{\mathbf{S}}=\mathbf{S}+\frac{1}{2}tI$, $\tilde{\mathbf{X}}=\mathbf{V}^T\mathbf{X}\mathbf{V}$, and $\tilde{\mathbf{C}}=\mathbf{V}^T\mathbf{C}\mathbf{V}$. So, basically, we have to perform a change of basis of the right hand side $\mathbf{C}$, and then transform back the computed solution to obtain $\mathbf{X}$.

Algorithm 5 (n-th time step)

```
Input: time $t$, timestep $\Delta t$, iteration number $n$, coefficient matrices $\tilde{\mathbf{C}}_2$, $\tilde{\mathbf{C}}_3$, $\tilde{\mathbf{M}}$,
       matrices from previous iterations $\mathbf{I}_{n-1}$, $\mathbf{J}_{n-1}$, $\left\{\tilde{\mathbf{M}}_k^\infty\right\}_{k=1,\ldots,n-1}$,
       Schur form $\mathbf{S}$ and orthogonal matrix of Schur vectors $\mathbf{V}$
Output: $\mathbf{I}_n$, $\mathbf{J}_n$, $\tilde{\mathbf{M}}_n^\infty$
  [5.1] $\mathbf{C}=-\sum_{k=1}^{n-1}\tilde{\mathbf{M}}_{n-k}^\infty\tilde{\mathbf{M}}_k^\infty$
  [5.2] $\mathbf{C}=\mathbf{C}-\tilde{\mathbf{C}}_2\left(\frac{\mathbf{J}_{n-1}}{\Delta t}+\mathbf{I}_{n-1}\right)-\left(\frac{\mathbf{J}_{n-1}}{\Delta t}+\mathbf{I}_{n-1}\right)\tilde{\mathbf{C}}_2^T$
  [5.3] $\mathbf{C}=\mathbf{C}+\frac{t^3}{6\Delta t}\tilde{\mathbf{C}}_3+\frac{t}{\Delta t}\left(\tilde{\mathbf{M}}-\mathbf{I}_{n-1}\right)$
  [5.4] $\tilde{\mathbf{C}}=\mathbf{V}^T\mathbf{C}\mathbf{V}$
  [5.5] $\tilde{\mathbf{S}}=\mathbf{S}+\frac{1}{2}tI$
  [5.6] Solve the Lyapunov equation (37) for $\tilde{\mathbf{X}}$
  [5.7] $\tilde{\mathbf{M}}_n^\infty=\mathbf{V}\tilde{\mathbf{X}}\mathbf{V}^T$
  [5.8] $\mathbf{J}_n=\mathbf{J}_{n-1}+\Delta t\mathbf{I}_{n-1}+\frac{\Delta t^2}{2}\tilde{\mathbf{M}}_n^\infty$
  [5.9] $\mathbf{I}_n=\mathbf{I}_{n-1}+\Delta t\tilde{\mathbf{M}}_n^\infty$
```

To finish the description of the algorithm, we mention an additional optimization (not shown in Algorithms 1–5) that reduces the number of required congruence transformations such as the ones just described, $\tilde{\mathbf{C}}=\mathbf{V}^T\mathbf{C}\mathbf{V}$. The cost of steps [5.4] and [5.7] is non-negligible (4 matrix-matrix multiplications per time step), but that can be avoided completely if the change of basis is performed outside the timestep loop. For this, the congruence transformation with $\mathbf{V}$ must be applied to matrices $\tilde{\mathbf{C}}_2$, $\tilde{\mathbf{C}}_3$, $\tilde{\mathbf{M}}$, $\mathbf{I}_0$, $\mathbf{J}_0$, and $\tilde{\mathbf{M}}_0^\infty$. Finally, the change of basis must be reversed, but this can be done in step [1.13] as $\mathbf{M}_n^\infty=\text{sparse}(\mathbf{W}\tilde{\mathbf{M}}_n^\infty\mathbf{W}^T)$ with $\mathbf{W}=\mathbf{L}\mathbf{V}$.

3.5 Numerical libraries

We will next describe the libraries used for the implementation of the time domain analysis. Figure 4 illustrates the main building blocks, with abstraction level increasing from bottom up. As mentioned before, the resulting code is hybrid, integrating sparse and dense computations. The upper dashed line separates libraries whose design philosophy is oriented to sparse computations (PETSc) from the rest. All parallel libraries follow the message-pasing programming paradigm, and are built on top of MPI [26]. Some of the software components are really sequential libraries (those below the lower dashed line) that are employed to perform local computations in the parallel algorithms. The arrows in the diagram indicate dependencies.

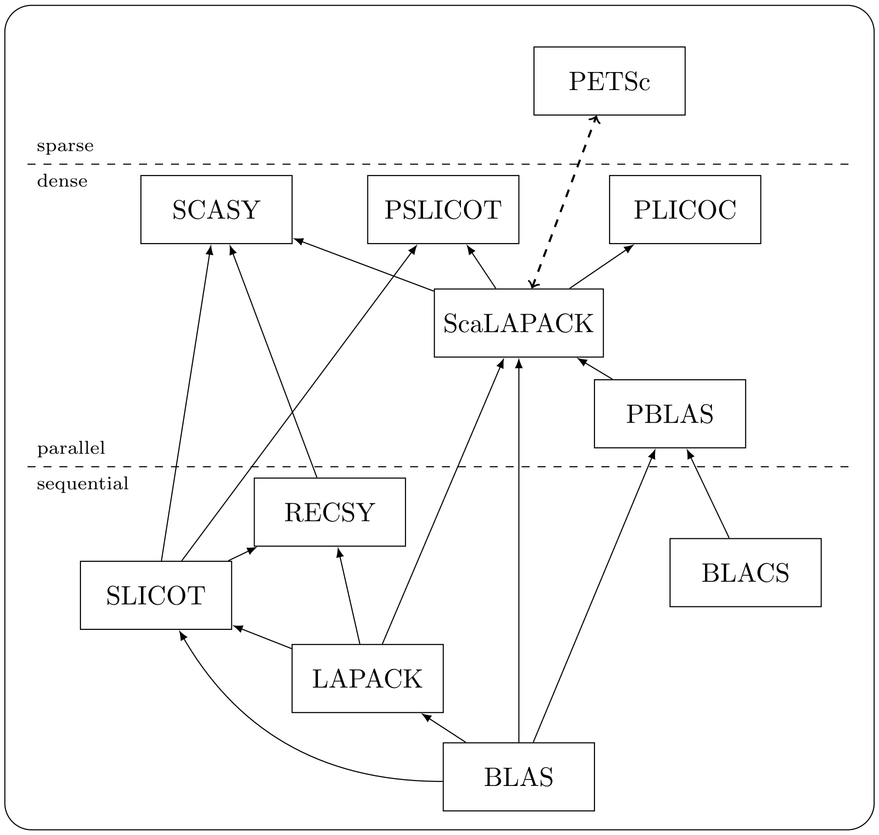


Figure 4: Software components employed for the implementation of the code

The heart of the hierarchy of libraries is ScaLAPACK, since most of the computation involves dense matrices. The interaction between the sparse and dense parts of the code is represented by the dashed arrow between ScaLAPACK and PETSc. This interface will be described in section 3.6.

LAPACK [1] is a well known sequential library for linear algebra computations with dense matrices. ScaLAPACK [12] is an attempt to provide the LAPACK functionality to users of parallel computers based on the message-passing paradigm. ScaLAPACK includes many useful subroutines for parallel linear algebra, and it is built on top of the BLACS [14] message passing interface and PBLAS, a parallel version of the BLAS. In ScaLAPACK, parallel matrices are stored in a 2-D block cyclic distribution fashion, to guarantee a good load balance in parallel computations. SCASY [16] provides ScaLAPACK-style algorithms for solving various standard and generalized Sylvester-type matrix equations. In particular, we have used subroutines for the (quasi-)triangular Lyapunov equation, which implement a block variant of the Bartels-Stewart (or Schur) method. Also, we have used some subroutines that were developed as part of the Parallel SLICOT [17], in particular a driver routine for the parallel Schur decomposition, as well as the PLICOC library [9] that includes a Newton solver for continuous-time algebraic Riccati equations as described in a previous subsection.

For the sparse computations we have used PETSc [5], a parallel framework for the numerical solution of problems arising in applications modeled by partial differential equations. It follows an object-oriented design to encapsulate data structures (such as parallel vectors an matrices) and solution algorithms (such as iterative linear solvers and preconditioners).

3.6 Implementation details

In this section we discuss several details related to the implementation of the codes. The first thing to consider is language interoperability. The developed codes are written in C++, whereas PETSc is programmed in C and ScaLAPACK and most of the rest of dense libraries are encoded in Fortran. Combining the three programming languages in a portable way requires a careful design of the interfaces. PETSc offers interfaces for C++ and Fortran programmers, as well as portable wrappers to LAPACK subroutines. We have extended this to also wrap subroutines from ScaLAPACK and the other libraries.

The code is structured as a PETSc application, in which the dense computations take place at some point, and therefore some calls have to be done to ScaLAPACK and related libraries. We now discuss how this integration has been implemented.

As mentioned before, the PETSc programming paradigm enforces encapsulation of data structures inside opaque objects. This is the case of `Mat`, the class intended for representing matrices in PETSc. A `Mat` object may have one of several internal representations, the most common being a parallel compressed sparse row format. The user may replace the internal data structure at run time, since application code uses an abstract interface that is independent from the actual data structure. Furthermore, PETSc provides simple extensibility mechanisms that allow, e.g., the creation of a new `Mat` format.

The integration of ScaLAPACK in a PETSc application amounts to providing a mechanism for accessing ScaLAPACK matrices in a PETSc style. Instead of the extensibility mechanism mentioned above, we have adopted a simpler scheme in which a “quasi-object” `ScaMat` is defined that mimics the behaviour of a `Mat`, but implements only the functionality that is strictly necessary for our application.

Table 1: Linear algebra operations implemented in the `ScaMat` interface.

Operation	Description	Library
`ScaMatScale`	$A \leftarrow \alpha A$	PBLAS
`ScaMatAdd`	$C \leftarrow \beta C + \alpha A$	PBLAS
`ScaMatAddTranspose`	$C \leftarrow \beta C + \alpha A^T$	PBLAS
`ScaMatSet`	$a_{ij} = \alpha \ (i \neq j)$, $a_{ii} = \beta$	ScaLAPACK
`ScaMatSetValue`	Set individual element a_{ij}	ScaLAPACK
`ScaMatTranspose`	$B \leftarrow A^T$	PBLAS
`ScaMatShift`	$A \leftarrow A + \alpha I$	ScaLAPACK
`ScaMatMatMult`	$C \leftarrow \beta C + \alpha AB$	PBLAS
`ScaMatNorm`	$r \leftarrow \|A\|$	ScaLAPACK
`ScaMatCholFactor`	$A = LL^T$ or $A = U^T U$	ScaLAPACK
`ScaMatCholCongruence`	$A \leftarrow L^{-1} A L^{-T}$	PBLAS
`ScaMatSchur`	$AV = VS$	PSLICOT
`ScaMatCARENewton`	$A^T X + XA - XGX + Q = 0$	PLICOC
`ScaMatLyapunov`	$AX + XA^T = C$	SCASY

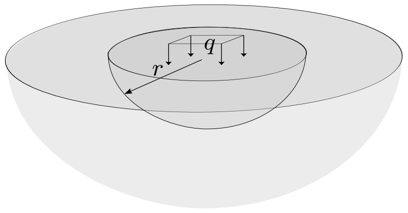
The `ScaMat` encapsulates all information necessary to handle a ScaLAPACK matrix, including the process grid context, the array descriptor, various sizes such as local and global dimensions, block sizes, and leading dimension, as well as the local array to store the elements of the 2-D block cyclic distributed matrix. Additionally, two boolean flags are included, one indicating whether the matrix is stored in symmetric format, and a second one to distinguish between storage in the upper or lower triangular parts.

We have implemented several basic object management functions for `ScaMat`, for creation, destruction, duplication and copy, visualization of information and matrix entries, and for naming. Also, we have functions to set the symmetry flag and to transform between symmetric and full (general) storage. The linear algebra operations that we employed are listed in Table 1. Additionally, we have implemented operations for inserting a PETSc `Mat` into a `ScaMat` (`ScaMatInsertMat`), and to convert a sparse `ScaMat` into a regular PETSc matrix based on a tolerance for nonzero elements (`ScaMatSparsify`).

4 Numerical results

In order to verify the implemented SBFEM code, we chose a simple settlement problem, consisting in the settlement of a flexible foundation under constant load. Using an infinite half space, the flexible foundation and its surrounding soil can be easily modeled. As already discussed in section 1, the problem has to be split into two separate domains. The near-field is represented by linear finite elements and the far-field by linear scaled boundary finite elements. The FEM/SBFEM coupling is implemented as illustrated in section 2.

Isotropic, homogeneous and fully linear elastic material can be described by three material parameters only: Young's modulus E, Poisson ratio ν, and density ρ. A semi-analytical solution is known for this problem [18]. This solution is valid for constant loads in time and space, so that the determined displacement is constant and time independent, which yields to a static solution.



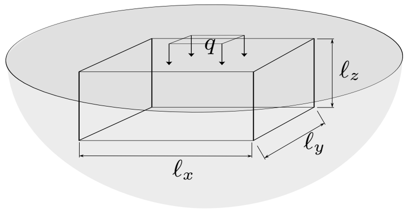


Figure 5: Infinite half space under constant area load q. (left) The computational domain is realized using a hemisphere. (right) The computational domain is realized using a cuboid.

Table 2: Hemisphere shaped meshes (HSM) with different numbers of degrees of freedom.

HSM	1	2	3	4	5	6	7	8	9
FEM	396	738	1314	3096	6030	9033	14655	19818	49155
SBFEM	123	219	291	480	843	1083	1515	1827	3603

To make sure that the implemented software is valid for different materials, two sets of parameters are defined.

The numerical model is implemented for time domain analysis so that the displacements are time-dependent. As the coupled FEM/SBFEM approach fulfills the radiation condition, the nodal displacements become constant after a certain time, whenever constant loads are applied. In those cases, the numerical results approximate the static semi-analytical solution. To show the accuracy of the method we use two different meshing strategies, which are discussed in detail in the following sections.

4.1 Hemisphere shaped mesh

We start the verification using a mesh, with the interface Γ of the infinite half space having a hemisphere-like shape. An area load of $q = 70 \left[\frac{kN}{m^2}\right]$ is applied on a square region of $152.4 \times 152.4 \left[m^2\right]$. The SBFEM scaling center is located in the middle of the load area. The distance between this center and all interface nodes at the boundary Γ is exactly $r = 190.5\,[m]$. Figure 5(left) depicts the setup of the problem.

In order to assess the accuracy of the method, the hemisphere-shaped mesh is refined several times, so that the geometry becomes smoother with each step of refinement. This also leads to an increasing number of degrees of freedom, as shown in Table 2.

In the first case, a set of material parameters is chosen as follows: $E = 37150.0 \left[\frac{kN}{m^2}\right]$, $\nu = 0.48\,[-]$ and $\rho = 1800.0 \left[\frac{kg}{m^3}\right]$ (set 1). This results in a speed of $c_p = 425.779 \left[\frac{m}{s}\right]$ for the longitudinal wave or p-wave, so that an adequate time step can be determined to $\Delta t = \frac{r}{30 c_p} = 0.0149\,[s]$ as it is suggested in [13]. Performing the simulation of the settlement problem for a period of $6.25\,[s]$, the HHT-α parameters are set to $\alpha = -\frac{1}{4}$, $\beta = \frac{(1-\alpha)^2}{4}$, $\gamma = \frac{1-2\alpha}{2}$, the time step Δt is chosen to $0.0125\,[s]$, and the number of time steps is set to 500.

Figure 6 (left) shows the time-dependent displacement of the settlement problem, which is normalized by the settlement $d\,(z) = 0.24800447\,[m]$ acquired from semi-analytic approach. The

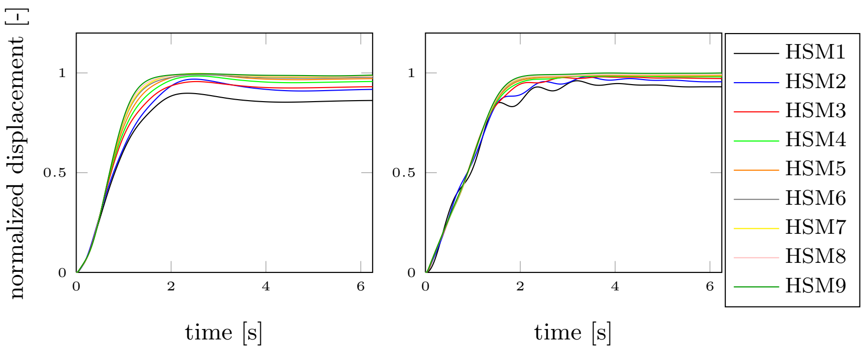


Figure 6: Normalized time-dependent displacement using meshes HSM 1 to 9 for two sets of material parameters: (left) set 1 (right) set 2.

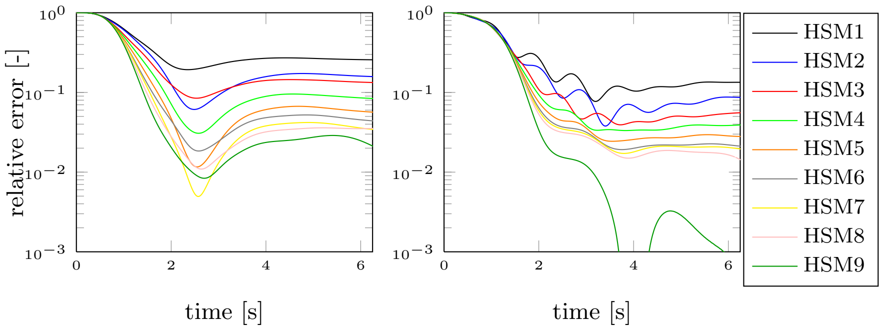


Figure 7: Relative error of time-dependent displacement using meshes HSM 1 to 9 for two sets of material parameters: (left) set 1 (right) set 2.

diagram clearly illustrates that the refinement of mesh leads to convergence of the semi-analytical solution. The relative error is given in Fig. 7 (left).

In the second case the following material parameters are used: $E = 21000.0\left[\frac{kN}{m^2}\right]$, $\nu = 0.13\,[-]$ and $\rho = 2100.0\left[\frac{kg}{m^3}\right]$ (set 2). Here, the speed of the p-wave is $c_p = 102.001\left[\frac{m}{s}\right]$ so that the adequate time step is approximated by $\Delta t = 0.06\,[s]$. The HHT-α parameters as well as the time step $\Delta t = 0.0125\,[s]$ are the same as in the previous example. The semi-analytic solution of the settlement for this set is $d\,(z) = 0.560443138\,[m]$. The normalized solution using the FEM/SBFEM coupled approach is shown in Fig. 6 (right), and the relative error compared to the semi-analytic solution in Fig. 7 (right). The results are similar to those obtained for the first case.

Table 3: Cuboid shaped meshes (CSM) with different numbers of degrees of freedom.

CSM	1	2	3	4	5
FEM	3042	20625	3042	3042	20625
SBFEM	-	-	1227	1227	4755

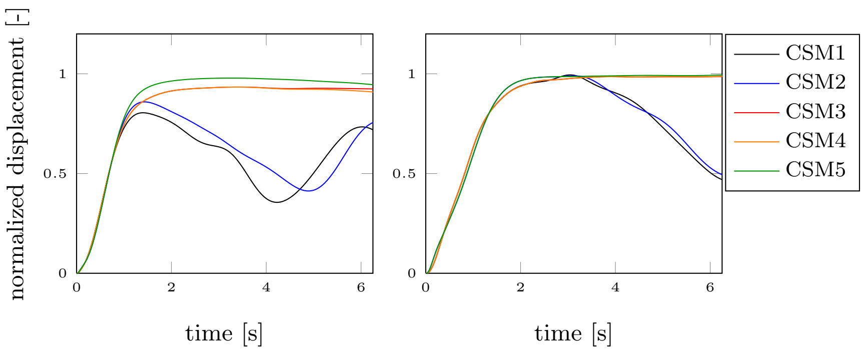


Figure 8: Normalized time-dependent displacement using meshes CSM 1 to 5 for two sets of material parameters: (left) set 1 (right) set 2.

4.2 Cuboid shaped mesh

As generating a hemisphere-like mesh is more complicated than creating a cuboid-shaped mesh (CSM), the settlement problem is done on a CSM as well. A sketch of the given problem is shown in Fig. 5 (right).

Like in the previous case, an area load of $q = 70\left[\frac{kN}{m^2}\right]$ is applied on a square region of $152.4 \times 152.4\left[m^2\right]$ with the SBFEM scaling center being located in the middle of this area. The dimensions of the mesh are set to $\ell_x = \ell_y = 457.2\,[m]$ and $\ell_z = 190.5\,[m]$, so that the smallest distance between scaling center and boundary Γ once more is $190.5\,[m]$.

The normalized displacements of the settlement are shown in Fig. 8. The first two graphs show the displacement of the settlement applying only FEM discretization (CSM1 and CSM2). In this case the boundary Γ is supported by simple roller boundary conditions which act perpendicular to the boundary. The free surface is not subjected to any boundary condition. As already discussed in section 1 this approach does not fulfill the radiation condition. It is easy to recognize that the displacement does not converge to the semi-analytic solution. The third graph (CSM3) shows the result using a modified version of a computer program called SIMILAR ("conSistent Infinitesimal finite-element cell Method - a fInite-eLement boundARy for modelling unbounded and bounded media") [32]. Using this software, the computation of a mesh with more degrees of freedom like CSM2 or CSM5 could not be done, due to the memory requirements. The last two graphs (CSM4 and CSM5) show the results using the newly developed SBFEM code. The normalized displacements of CSM3 and CMS4 fit in both cases very well. The mesh with the highest resolution (CSM5) converges to the semi-analytic solution, as expected.

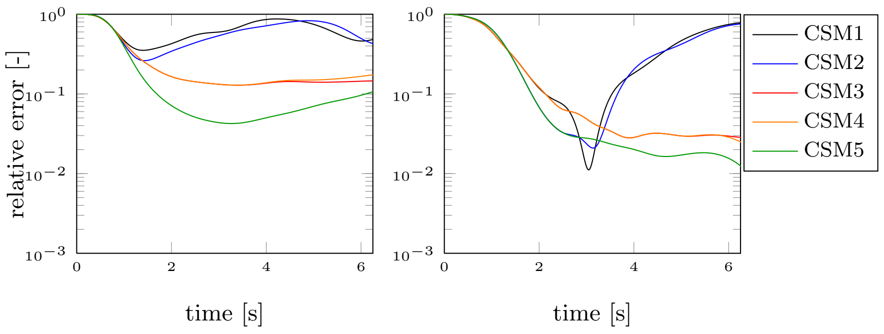


Figure 9: Relative error of time-dependent displacement using meshes CSM 1 to 5 for two sets of material parameters: (left) set 1 (right) set 2.

5 Evaluation of parallel performance

In this section we report the parallel performance of the code. The tests have been carried out on a cluster of 28 nodes, each with 2 Opteron-240 processors at 1.4 GHz and 3 GB of RAM, linked via a Myrinet-2000 interconnect. We have used one MPI rank per processor, fully populating one node before adding a new one (thus, for example, experiments with 2 processes employ the two CPUs in a single node). The parallel performance is analyzed in terms of both strong and weak scaling. For strong scaling, time measurements are done considering a constant problem size and increasing number of processors. For weak scaling, the problem size grows proportionally to the number of processors.

5.1 Strong scaling

Table 4 shows the execution times for different number of processors in the solution of problem CSM3, described in the previous section. The number of unknowns for this problem in the SBFEM analysis is 1,227. To evaluate the parallel performance the analysis has been limited to 100 time steps (only 100 $\mathbf{M}_n^\infty$ matrices need to be stored). For this small example we cannot expect very good parallel performance, because the local problem size gets too small with increasing number of processes. This eventually leads to a negligible computation time while the overhead of communication is ever increasing.

From the times corresponding to 1 processor, we can observe that more than 99.5% of the time is spent in the time domain analysis (shaded part in the flowchart of Fig. 3, which corresponds to Algorithm 1). The computation of the first time step is dominated by the Riccati solver (Algorithm 3), and the time required for subsequent time steps (Algorithm 5) represents the major part of time consumption. This latter time accounts mainly for the convolution integral and the Lyapunov solver. The remaining time belonging to the time domain analysis (not shown in Table 4) is mainly due to the computation of the Schur factorization and the coefficient matrices.

For better evaluation of the parallel performance of each part of the computation, Fig. 10

Table 4: Measured computational time (in seconds) for different number of processors (p) corresponding to the overall computation, the time domain analysis (Algorithm 1), the Riccati solver (Algorithm 3), and the n-th time step (Algorithm 5).

p	Overall	Algorithm 1	Algorithm 3	Algorithm 5
1	14,617	14,551	343	13,920
3	6,163	6,136	196	5,743
6	3,279	3,262	106	3,022
8	2,723	2,707	93	2,495
12	1,925	1,911	74	1,743
15	1,725	1,713	68	1,549
20	1,482	1,469	68	1,321
24	1,375	1,363	57	1,237
30	1,042	1,031	54	914
35	1,044	1,028	57	911
48	902	889	47	785
56	885	870	50	762

displays the parallel speedup, which is defined as the ratio between time with one and p processes,

$$S_p = \frac{T(1)}{T(p)}, \tag{38}$$

representing the gain factor when the code is run with p processors. The plot shows a steady increasing trend of the speedup, although far from the theoretical maximum (line "Theory" in Fig. 10). Here, the maximum achieved speedup for the time domain analysis (Algorithm 1) is about 20 with 56 processes.

The parallel efficiency

$$E_p = \frac{T(1)}{p \cdot T(p)}, \tag{39}$$

gives an indication of how well the code is utilizing the available processors. In this case, the efficiency falls to, approximately, 35% for the largest number of processors. Such a small value can be attributed to the small problem size, so we repeat the analysis with the larger CSM5 ($n = 4755$).

Due to much larger memory requirements for CSM5, we decrease the number of time steps to 5 in order to avoid disk swap due to insufficient physical memory when run with 1 or 2 processors. We observe a speedup for 24 and 48 processors of 14.82 and 17.47 and a parallel efficiency of 62% and 36%, respectively. This indicates that speedup and parallel efficiency increase when the problem becomes larger.

5.2 Weak scaling

We now proceed with a weak scaling analysis to prove applicability of this approach to real problems with a large number of unknowns. In this case, the problem size increases proportionally to the number of processes to ensure that the local problem size is roughly constant for each processor.

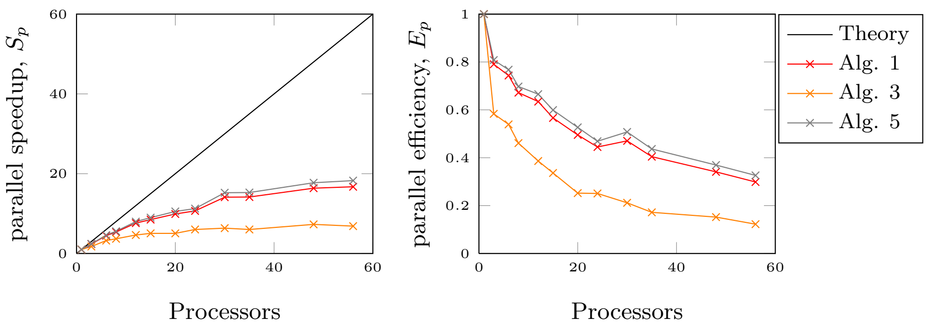


Figure 10: (left) Parallel speedup, (right) parallel efficiency corresponding to the main parts of the computation.

In each process a submatrix with $N \approx 1,200,000$ entries is tackled (the number varies slightly due to geometric discretization). To simplify the exposition, we assume that the number of processes p is a perfect square, so that processes are arranged as a $\sqrt{p} \times \sqrt{p}$ grid, and the local submatrix has size $n \times n$, where $n = \sqrt{N}$.

Suppose $T(1)$ represents the time associated to a problem of size n on 1 process. In the weak scaling analysis, we study the time for a problem of size $n\sqrt{p}$ with p processes, which will be denoted as $T'(p)$. The (scaled) speedup would be defined as before, that is, the ratio between $T'(1)$ and $T'(p)$, but since $T'(1)$, the time to solve a problem of size $n\sqrt{p}$ with 1 process, is not available, we approximate it in terms of $T(1)$. Since the cost of the algorithm is cubic in the dimension of the matrix, for a size of $n\sqrt{p}$ we have $O(n^3 \cdot p\sqrt{p})$. Then the scaled speedup can be defined in this case as

$$S'_p = \frac{p\sqrt{p} \cdot T(1)}{T'(p)}, \tag{40}$$

and the corresponding scaled efficiency is

$$E'_p = \frac{\sqrt{p} \cdot T(1)}{T'(p)}. \tag{41}$$

Both quantities are depicted in Figure 11.

For programs with good scaling characteristics the parallel efficiency is constant. In our case, the graphs display close-to-horizontal lines for 3 to 56 processors, which indicates that the developed code will scale well for large-scale applications. The parallel efficiency is much higher when using only 1 or 2 nodes. This is likely due to the non existent or more effective communication while running on one compute node.

6 Discussion and concluding remarks

The methodology and results described in this paper show that, in practice, parallel computing can provide an answer to the high computational cost resulting from the application of SBFEM to very large scaled engineering problems. The parallel method that we designed to deal with

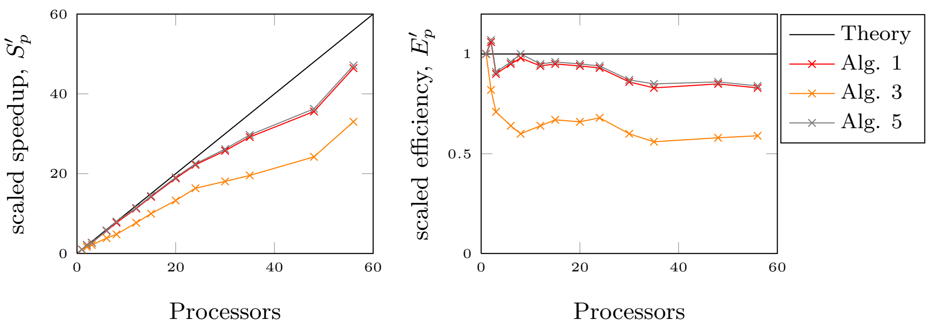


Figure 11: (left) Scaled speedup, (right) scaled efficiency corresponding to the main parts of the computation.

these problems makes extensive use of external numerical libraries for linear algebra computations: due to the nature of the computation, a hybrid sparse-dense program is required, which adds a significant programming complexity as it requires the integration of numerical libraries of different nature, and the use of the appropriate data structures in each case. The proposed solution mixes PETSc and ScaLAPACK functionality, resulting in an efficient and reasonably scalable code.

The proposed algorithm exhibits a good numerical behaviour as the numerical test cases evaluated in section 4 feature convergence to the semi-analytical solution, as expected. This demonstrates that, in practice, when run on a moderately-sized cluster, the SBFEM method can yield the solution not only to academic benchmarks, but also to real problems.

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 User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992.

[2] H. Antes and C. Spyrakos. Soil-structure interaction. In D.E. Beskos and S.A. Anagnotopoulos, editors, *Computer Analysis and Design of Earthquake Resistant Structures*, page 271. Computational Mechanics Publications, Southampton, 1997.

[3] Daniel Appelö and Tim Colonius. A high-order super-grid-scale absorbing layer and its application to linear hyperbolic systems. *Journal of Computational Physics*, 228(11):4200–4217, 2009.

[4] R. J. Astley. Infinite elements for wave problems: a review of current formulations and a assessment of accuracy. *International Journal for Numerical Methods in Engineering*, 49(7):951–976, 2000.

[5] Satish Balay, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matt Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.1, Argonne National Laboratory, 2010.

[6] P. Benner. *Contributions to the Numerical Solution of Algebraic Riccati Equations and Related Eigenvalue Problems*. Dissertation, Fak. f. Mathematik, TU Chemnitz–Zwickau, Chemnitz, FRG, 1997.

[7] P. Benner. Numerical solution of special algebraic Riccati equations via an exact line search method. In *Proc. European Control Conf. ECC 97, Paper 786*, Waterloo (B), 1997. BELWARE Information Technology.

[8] P. Benner, R. Byers, E.S. Quintana-Ortí, and G. Quintana-Ortí. Solving algebraic Riccati equations on parallel computers using Newton's method with exact line search. *Parallel Comput.*, 26(10):1345–1368, 2000.

[9] P. Benner, E. S. Quintana-Ortí, and G. Quintana-Ortí. Solving linear-quadratic optimal control problems on parallel computers. *Optimization Methods and Software*, 23(6):879–909, 2008.

[10] P. Benner and E.S. Quintana-Ortí. Solving stable generalized Lyapunov equations with the matrix sign function. *Numer. Alg.*, 20(1):75–100, 1999.

[11] P. Bettess. *Infinite Elements*. Penshaw Press, Sunderland, U.K., 1992.

[12] 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*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[13] Robert Borsutzky. *Braunschweiger Schriften zur Mechanik - Seismic Risk Analysis of Buried Lifelines*, volume 63. Mechanik-Zentrum Technische Universität Braunschweig, 2008.

[14] J. J. Dongarra and R. C. Whaley. LAPACK working note 94: A user's guide to the BLACS v1.1. Technical Report UT-CS-95-281, Department of Computer Science, University of Tennessee, 1995.

[15] B. Engquist and A. Majda. Absorbing boundary conditions for the numerical simulation of waves. *Mathematics of Computation*, 31(139):629–651, 1977.

[16] Robert Granat and Bo Kågström. Algorithm 904: The SCASY library – parallel solvers for Sylvester-type matrix equations with applications in condition estimation, part II. *ACM Transactions on Mathematical Software*, 37(3):33:1–33:4, 2010.

[17] D. Guerrero, V. Hernández, and J. E. Román. Parallel SLICOT model reduction routines: The Cholesky factor of Grammians. In *Proceedings of the 15th Triennal IFAC World Congress*, Barcelona, Spain, 2002.

[18] M. E. Harr. *Foundations of Theoretical Soil Mechanics*. McGraw-Hill Book Company, New York, 1966.

[19] H. Hilbert, T. Hughes, and R. Taylor. Improved numerical dissipation for time integration algorithms in structural dynamics. *Earthquake Engineering & Structural Dynamics*, 5:283, 1977.

[20] D.L. Kleinman. On an iterative technique for Riccati equation computations. *IEEE Trans. Automat. Control*, AC-13:114–115, 1968.

[21] L. Lehmann. *Wave Propagation in Infinite Domains.* Springer, Berlin / Heidelberg, 2006.

[22] L. Lehmann, S. Langer, and D. Clasen. Scaled boundary finite element method for acoustics. *Journal of Computational Acoustics*, 14(4):489–506, 2006.

[23] Z. P. Liao and H. L. Wong. A transmitting boundary for the numerical simulation of elastic wave propagation. *Soil Dynamics and Earthquake Engineering*, 3(4):174–183, 1984.

[24] J. Lysmer and R. L. Kuhlmeyer. Finite dynamic model for infinite media. *Journal of Engineering Mechanics*, 95:859–875, 1969.

[25] Konstantin Meskouris, Klaus-G. Hinzen, Christoph Butenweg, and Michael Mistler. *Bauwerke und Erdbeben - Grundlagen - Anwendung - Beispiele.* Vieweg+Teubner Verlag, Wiesbaden, 2007.

[26] MPI Forum. The message passing interface (MPI) standard. http://www.mcs.anl.gov/mpi, 1994.

[27] N. Newmark. A method of computation for structural dynamics. *Journal of Engineering Mechanics Division*, 85:67, 1959.

[28] Christan Petersen. *Dynamik der Baukonstruktionen.* Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 2000.

[29] J.D. Roberts. Linear model reduction and solution of the algebraic Riccati equation by use of the sign function. *Internat. J. Control*, 32:677–687, 1980.

[30] M. Schauer and L. Lehmann. Large scale simulation with scaled boundary finite element method. *Proceedings in Applied Mathematics and Mechanics*, 9:103 – 106, 2009.

[31] J. Wolf. *The Scaled Boundary Finite Element Method.* John Wiley & Sons, Chichester, 2003.

[32] J. Wolf and C. Song. *Finite-Element Modelling of Unbounded Media.* John Wiley & Sons, Chichester, 1996.