Román Moltó, JE.; Vasconcelos, PB.; Nunes, AL. (2013). Eigenvalue computations in the context of data-sparse approximations of integral operators. Journal of Computational and Applied Mathematics. 237(1):171-181. doi:10.1016/j.cam.2012.07.021.

Eigenvalue Computations in the Context of Data-Sparse Approximations of Integral Operators

J. E. Roman*

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

P. B. Vasconcelos

Centro de Matemática, and Faculdade Economia, Universidade Porto, R. Dr. Roberto Frias s/n, 4200-464 Porto, Portugal

A. L. Nunes

Instituto Politécnico do Cávado e do Ave, Av. Dr. Sidónio Pais 222, 4750-333 Barcelos, Portugal

Abstract

In this work, we consider the numerical solution of a large eigenvalue problem resulting from a finite rank discretization of an integral operator. We are interested in computing a few eigenpairs, with an iterative method, so a matrix representation that allows for fast matrix-vector products is required. Hierarchical matrices are appropriate for this setting, and also provide cheap LU decompositions required in the spectral transformation technique. We illustrate the use of freely available software tools to address the problem, in particular SLEPc for the eigensolvers and HLib for the construction of $\mathcal{H}$-matrices. The numerical tests are performed using an astrophysics application. Results show the benefits of the data-sparse representation compared to standard storage schemes, in terms of computational cost as well as memory requirements.

Keywords: iterative eigensolvers, integral operator, hierarchical matrices,

*Corresponding author

Email addresses: jroman@dsic.upv.es (J. E. Roman), pjv@fep.up.pt (P. B. Vasconcelos), anunes@ipca.pt (A. L. Nunes)

numerical libraries
2010 MSC: 65F15, 65F50, 65R20, 65Y20

1. Introduction

Computing a few eigenvalues (and corresponding eigenvectors) of matrices arising in large-scale scientific computing applications can be challenging. In the last decades, very effective iterative methods have been devised to approximate the eigenspace associated to any part of the spectrum. Examples of such techniques are restarted Krylov methods and preconditioned eigensolvers such as Jacobi-Davidson [1]. Also, these methods are progressively taking shape as high-quality implementations in software libraries such as SLEPc, the Scalable Library for Eigenvalue Problem Computations [2], thus enabling application programmers to cope with challenging problems coming from a wide range of applications.

Libraries such as SLEPc try to make problems computationally tractable by combining two main ingredients: (i) exploiting sparsity of the matrices, and (ii) exploiting parallelism.

Sparsity of matrices is a desirable property that appears in the context of partial differential equations with standard discretization techniques such as the finite element method. This situation is very common in practice, and allows iterative methods to be competitive by benefiting from the cheap, linear-cost matrix-vector products. However, there are cases where the problem is formulated as an integral equation, either from the very nature of the problem or from a partial differential equation being formulated in boundary integral form. In these cases, sparsity of matrices is not guaranteed, so in principle full (dense) storage must be used, with the consequent blow-up in computational cost. The concept of hierarchical matrix (or $\mathcal{H}$-matrix) was introduced by Hackbusch and co-workers [3, 4] with the aim of providing a cheap representation for such class of matrices. The main idea is to build a hierarchical decomposition of the matrix in blocks, where most of the off-diagonal blocks are represented as the product of two low rank matrices. The result is a data-sparse storage scheme, where the memory requirement for a square matrix of dimension n is only $\mathcal{O}(n \log n)$. By reducing the computational cost to linear-logarithmic complexity, iterative schemes are attractive again. Furthermore, the $\mathcal{H}$-matrix representation also allows for inexpensive computation of some operations that would otherwise be prohibitive, such as matrix inversion.

In this contribution, we concentrate on the use of $\mathcal{H}$-matrix representation in the context of eigenvalue computations, similarly to [5]. Our main focus here is on using freely available software tools, in particular SLEPc for the eigensolvers and HLib [6, 7] for the $\mathcal{H}$-matrix representation. Numerical experiments are performed on a weakly singular integral operator, whose discretization leads to a matrix that can be large.

Regarding parallelization, the other important ingredient mentioned before, it becomes less critical for the case of $\mathcal{H}$-matrix computations, due to the reduced computational cost (compared to dense storage). However, it is still desirable whenever the problem size grows considerably. In this work, we will make use of parallelization only naively.

We consider an eigenvalue problem, issued from an integral formulation of a transfer problem in stellar atmospheres [8, 9], with operator $T : X \to X$, $X = L^1([0, \tau^\star])$, defined by

$$(Tx)(\tau) := \int_0^{\tau^\star} g(\tau, \sigma) x(\sigma) d\sigma, \quad \tau \in [0, \tau^\star], \tag{1}$$

where $\tau^\star$ is finite, and with kernel

$$g(\tau, \sigma) := \frac{\varpi}{2} E_1(|\tau - \sigma|). \tag{2}$$

The kernel, which is weakly singular, is defined through the first exponential-integral function, the first of the sequence of functions [10]

$$E_\nu(\tau) := \int_1^\infty \frac{\exp(-\tau\mu)}{\mu^\nu} d\mu, \quad \tau > 0, \nu \geq 0, \tag{3}$$

and depends on the albedo, $\varpi \in [0, 1]$, considered in this work as a constant.

To solve the eigenproblem

$$T\varphi = \lambda\varphi \tag{4}$$

with $\lambda \neq 0$ and $\varphi \neq 0$, $\varphi \in X$, we consider a class of operator approximations for which the range is a finite dimensional subspace of X. A finite rank approximation T_n of T is constructed by considering a family of grids $(\tau_{n,j})_{j=0}^n$ on $[0, \tau^\star]$. For $x \in X$ we set

$$\langle x, e^*_{n,j} \rangle := \frac{1}{\tau_{n,j} - \tau_{n,j-1}} \int_{\tau_{n,j-1}}^{\tau_{n,j}} x(\sigma) d\sigma,$$

where $e_{n,j} = 1$ if $\tau \in]\tau_{n,j-1}, \tau_{n,j}[$ and 0 otherwise. A bounded n-rank projection onto the subspace $X_n = \text{span}\{e_{n,j} : j = 1, ..., n\}$ is defined by

$$\pi_n x := \sum_{j=1}^{n} \langle x, e_{n,j}^* \rangle e_{n,j},$$

and

$$T_n x = \pi_n T x.$$

To obtain a matrix representation of

$$T_n \varphi_n = \theta_n \varphi_n \tag{5}$$

the spectral problem for T_n is then reduced to an $n \times n$ matrix eigenproblem

$$A_n x_n = \theta_n x_n, \tag{6}$$

where $A_n(i, j) := \langle T e_{n,j}, e_{n,i}^* \rangle$. See [11] for additional details.

In section 2 we discuss several strategies for storing the discretized operator A_n, including the $\mathcal{H}$-matrix representation. Section 3 provides a brief overview of iterative solution methods for the partial eigenvalue problem. In section 4 we provide details specific to our implementation, and in section 5 we show the results of some numerical experiments.

2. Matrix Representations of the Discretized Operator

In this section, we discuss several strategies for representing matrix A_n. We start by showing the formula used in the standard dense storage, and then we move to the data-sparse representation, which can be implemented in various flavours.

2.1. Explicit Computation and Storage of Matrix Elements

The generation of A_n for large n is very time consuming, being often more expensive than the computation of the solution of the eigenvalue problem itself. For the present kernel,

$$A_n(i, j) = \frac{\varpi}{2(\tau_{n,i} - \tau_{n,i-1})} \int_{\tau_{n,i-1}}^{\tau_{n,i}} \int_{\tau_{n,j-1}}^{\tau_{n,j}} E_1 \left(|\tau - \sigma| \right) d\sigma d\tau \tag{7}$$

$$= \begin{cases} \frac{\varpi}{2(\tau_{n,i}-\tau_{n,i-1})}(-E_3(|\tau_{n,i}-\tau_{n,j}|)+E_3(|\tau_{n,i-1}-\tau_{n,j}|)+ \\ \quad +E_3(|\tau_{n,i}-\tau_{n,j-1}|)-E_3(|\tau_{n,i-1}-\tau_{n,j-1}|)), & i \neq j \\ \varpi[1+\frac{1}{\tau_{n,i}-\tau_{n,i-1}}(E_3\left(\tau_{n,i}-\tau_{n,i-1}\right)-\frac{1}{2})], & i=j \end{cases} \tag{8}$$

For each (i, j), four evaluations of the function E_3 are required. There is a clear decay in magnitude away from the diagonal, depending on $\tau^\star$ and on n: for constant values of the former, smaller values of the latter imply faster decay from the diagonal. The idea of zeroing out all entries with magnitude less than a certain tolerance to avoid working with dense matrix storage was investigated in [12], and a theoretical treatment validated this approach for a required precision [13]. Nevertheless, this strategy requires the computation of every matrix entry in order to evaluate its magnitude, since an a priori determination of the maximum bandwidth is not possible.

Apart from the high generation cost, the main drawback of dense (or banded) storage is that operations such as matrix-vector products are expensive. As discussed previously, data-sparse representation such as the $\mathcal{H}$-matrix scheme may be useful to tackle this kind of problems.

2.2. Data-Sparse Representation

Grounded on the notion of distance between two subsets of an index set, an admissibility condition is defined. From this condition test, it is decided whether certain blocks of the matrix can be computed approximately and stored in a special format, leading to less memory and computational costs.

Let $\mathcal{I} = \{1, ..., n\}$ be the set of indices of the basis functions $e_{n,i}$, and let t and s be two subsets of $\mathcal{I}$ where $\alpha = \bigcup_{i\in t} \text{supp } e_{n,i}$ and $\beta = \bigcup_{j\in s} \text{supp } e_{n,j}$ are the corresponding domains.

Beginning with the root of the tree $T_{\mathcal{I}\times\mathcal{I}}$, that is $\mathcal{I} \times \mathcal{I}$, it is split up in four and an *admissibility condition* (ensuring exponential convergence) is tested (see [14]):

$$\text{diam}(\alpha) \leq \eta \, \text{dist}(\alpha, \beta), \tag{9}$$

for $\eta > 0$ fixed. When a block is admissible, that is, the corresponding domain $\alpha \times \beta$ is admissible, the division is stopped and a low rank approximation is used to represent the block; otherwise, the division process continues recursively up to a given minimum size.

A matrix is stored in $\mathcal{H}$-matrix representation if, given a block cluster tree for an index set $\mathcal{I}$ of cardinality k, the sub-matrices corresponding to admissible leaves are stored in the factorized form $AB^T \in \mathbb{R}^{t\times s}$ with rank

at most k, $A \in \mathbb{R}^{t\times k}$ and $B \in \mathbb{R}^{s\times k}$. The sub-blocks of inadmissible leaves are stored as dense matrices. This is accomplished by replacing the kernel $g(\tau, \sigma)$ by a degenerate approximation $\tilde{g}(\tau, \sigma)$, such that the integration with respect to the different variables is segregated

$$\tilde{g}(\tau, \sigma) := \sum_{\rho=0}^{k-1} f_\rho(\tau) h_\rho(\sigma). \tag{10}$$

In this work, the process of building the cluster tree is undertaken by HLib [6, 7], which, as already mentioned, enables matrix operations of almost linear complexity, being therefore particularly adequate for large dimensional problems. Alternatively, AHMED (Another software library on Hierarchical Matrices for Elliptic Differential equations) could be used. A complete reference for $\mathcal{H}$-matrices as well as for this library is [15].

With the $\mathcal{H}$-matrix representation, it is possible to realize common computations with linear-polylogarithmic complexity rather than quadratic or cubic cost. For instance, the multiplication of an $n \times n$ $\mathcal{H}$-matrix by a vector requires about $4n \log_2 n$ floating-point operations, the (approximate) LU decomposition about $6n \log_2^2 n$ operations, and $2n \log_2 n$ operations for the backsolves. See [3, 4] for additional details on $\mathcal{H}$-matrix arithmetic.

Degenerate approximations can be built in different ways, for instance, based on polynomial interpolation or Taylor series expansion. Another approach consists in computing a low rank approximation from an explicitly built matrix block by means of a singular value decomposition. We next describe these three approaches in turn.

2.2.1. SVD

A singular value decomposition (SVD) on each admissible block, with already generated entries, can be used to preserve the most valuable information and discard the rest. The resulting rank-k approximation can be expressed as

$$AB^T = \sum_{\rho=1}^{k} u_\rho \sigma_\rho v_\rho^T, \tag{11}$$

where σ_ρ are the singular values (in descending order of magnitude), and u_ρ and v_ρ are the corresponding left and right singular vectors, respectively. The rank, k, can be chosen to be a fixed value, or alternatively to be set dynamically on each block, based on a prescribed tolerance ϵ. In the latter

case, the condition $\sigma_k > \epsilon \geq \sigma_{k+1}$ holds. In computational terms, the cost to obtain the $\mathcal{H}$-matrix is high since the entries of every admissible block must be explicitly generated first, in the present case through (8), prior to the singular value decomposition. Nevertheless, all the computations done afterwards can be performed cheaply.

2.2.2. Series Expansion

For the considered problem, the derivatives of the kernel function can be derived analytically (elegantly, by means of a recursive definition [10]). This allows for the use of truncated Taylor series as a degenerate approximation for the kernel of the integral operator.

For the inadmissible blocks, the entries of the matrix are computed according to (8).

If the admissibility condition holds, the kernel $g(\tau, \sigma)$ is replaced by its truncated Taylor series, according to (10) with $f_\rho(\tau) = (\tau - \tau_0)^\rho$ and $h_\rho(\sigma) = \frac{1}{\rho!}\partial_\tau^\rho g(\tau_0, \sigma)$. Here, we take τ_0 as the midpoint of the α interval.

The matrix entries $G_{ij} = \int_0^{\tau^*}\int_0^{\tau^*} e_{n,i} g(\tau, \sigma) e_{n,j} d\sigma d\tau$ for $(i, j) \in t \times s$ will be approximately computed by:

$$\begin{aligned}
\tilde{G}_{ij} &= \int_0^{\tau^*}\int_0^{\tau^*} e_{n,i}\tilde{g}(\tau, \sigma) e_{n,j} d\sigma d\tau \\
&= \int_0^{\tau^*}\int_0^{\tau^*} e_{n,i} \sum_{\rho=0}^{k-1} f_\rho(\tau) h_\rho(\sigma) e_{n,j} d\sigma d\tau \\
&= \sum_{\rho=0}^{k-1} \underbrace{\int_0^{\tau^*} e_{n,i} f_\rho(\tau) d\tau}_{A_{i\rho}} \underbrace{\int_0^{\tau^*} e_{n,j} h_\rho(\sigma) d\sigma}_{B_{j\rho}}
\end{aligned}$$

The sub-matrix $\tilde{G}|_{t\times s} := AB^T$, where $A \in \mathbb{R}^{t\times\{0,\ldots,k-1\}}$, $B \in \mathbb{R}^{s\times\{0,\ldots,k-1\}}$ and $A_{i\rho} = \int_{\tau_{n,i-1}}^{\tau_{n,i}} f_\rho(\tau) d\tau$ and $B_{j\rho} = \int_{\tau_{n,j-1}}^{\tau_{n,j}} h_\rho(\sigma) d\sigma$, has rank at most k.

Now, we will derive analytical expressions for $A_{i\rho}$ and $B_{j\rho}$.

Proposition 1. *The entries of matrix A can be computed as*

$$A_{i\rho} = \frac{1}{h_{n,i}} \int_{\tau_{n,i-1}}^{\tau_{n,i}} (\tau - \tau_0)^\rho d\tau = \frac{1}{h_{n,i}} \left(\frac{(\tau_{n,i} - \tau_0)^{\rho+1}}{\rho + 1} - \frac{(\tau_{n,i-1} - \tau_0)^{\rho+1}}{\rho + 1} \right), \tag{12}$$

where $h_{n,i} = \tau_{n,i} - \tau_{n,i-1}$.

Proof. Immediate. □

Before deducing the formula for the entries $B_{j\rho}$ of the matrix B, one needs to obtain the expression for $\partial_\tau^\rho g(\tau_0, \sigma)$.

Proposition 2. *The partial derivatives of the kernel $g(\tau, \sigma)$ of (2) with respect to the first argument are*

$$\partial_\tau^\rho g(\tau,\sigma) = \begin{cases} -\frac{\varpi}{2}\sum_{k=1}^{\rho}(-1)^{k-1}\frac{(\rho-1)!}{(\rho-k)!}\frac{e^{\tau-\sigma}}{(\tau-\sigma)^k}, & \sigma > \tau \\ -\frac{\varpi}{2}\sum_{k=1}^{\rho}(-1)^{\rho-1}\frac{(\rho-1)!}{(\rho-k)!}\frac{e^{-\tau+\sigma}}{(\tau-\sigma)^k}, & \sigma < \tau \end{cases} . \tag{13}$$

Proof. We know from [10, §5] that

$$E_1(z) = \int_1^\infty \frac{e^{-zt}}{t}dt \qquad \text{and} \qquad \frac{d}{dz}E_1(z) = -E_0(z) = -\frac{e^{-z}}{z}.$$

Starting with the first derivative $\partial_\tau^1 g(\tau, \sigma)$,

$$\partial_\tau g(\tau,\sigma) = \frac{\varpi}{2}\partial_\tau E_1(|\tau-\sigma|) = \begin{cases} -\frac{\varpi}{2}\frac{e^{-(\tau-\sigma)}}{\tau-\sigma}, & \sigma < \tau \\ -\frac{\varpi}{2}\frac{e^{-(-\tau+\sigma)}}{\tau-\sigma}, & \sigma > \tau \end{cases} = -\frac{\varpi}{2}\frac{e^{-|\tau-\sigma|}}{\tau-\sigma} \ (\text{for } \tau \neq \sigma).$$

1. For $\sigma > \tau$, after some initial calculations, we obtain

$$\partial_\tau^2 g(\tau,\sigma) = -\frac{\varpi}{2}\left(\frac{e^{\tau-\sigma}}{\tau-\sigma} - \frac{e^{\tau-\sigma}}{(\tau-\sigma)^2}\right),$$
$$\partial_\tau^3 g(\tau,\sigma) = -\frac{\varpi}{2}\left(\frac{e^{\tau-\sigma}}{\tau-\sigma} - \frac{2e^{\tau-\sigma}}{(\tau-\sigma)^2} + \frac{2e^{\tau-\sigma}}{(\tau-\sigma)^3}\right),$$

   and, generalizing, we reach the expression for any value of ρ

$$\partial_\tau^\rho g(\tau,\sigma) = -\frac{\varpi}{2}\sum_{k=1}^{\rho}(-1)^{k-1}\frac{(\rho-1)!}{(\rho-k)!}\frac{e^{\tau-\sigma}}{(\tau-\sigma)^k}.$$

2. For $\sigma < \tau$, analogously to the previous case, the following expression is obtained

$$\partial_\tau^\rho g(\tau,\sigma) = -\frac{\varpi}{2}\sum_{k=1}^{\rho}(-1)^{\rho-1}\frac{(\rho-1)!}{(\rho-k)!}\frac{e^{-\tau+\sigma}}{(\tau-\sigma)^k}.$$

□

With the above result we can readily obtain the expression for the entries of matrix B, by evaluating the derivatives in $\tau = \tau_0$.

Proposition 3. *The entries of matrix B can be computed as*

$$B_{j\rho} = \begin{cases} \frac{\varpi}{2} \sum_{k=1}^{\rho} \frac{1}{\rho(\rho-k)!} \left[\Gamma\left(1-k, -\tau_0 + \tau_{n,j-1}\right)\right. & \\ \qquad \left. -\Gamma\left(1-k, -\tau_0 + \tau_{n,j}\right)\right], & \tau_{n,j-1} > \tau_0,\ \tau_{n,j} > \tau_0 \\ -\frac{\varpi}{2} \sum_{k=1}^{\rho} \frac{(-1)^{\rho-1}}{\rho(\rho-k)!} \left[\Gamma\left(1-k, \tau_0 - \tau_{n,j}\right)\right. & \\ \qquad \left. -\Gamma\left(1-k, \tau_0 - \tau_{n,j-1}\right)\right], & \tau_{n,j-1} < \tau_0,\ \tau_{n,j} < \tau_0 \end{cases}$$

Proof. First recall the incomplete Gamma function

$$\Gamma(a, x) = \int_x^{\infty} \frac{e^{-t}}{t^{1-a}} dt.$$

1. For $\tau_{n,j-1} > \tau_0,\ \tau_{n,j} > \tau_0$:

$$\begin{aligned} B_{j\rho} &= \int_{\tau_{n,j-1}}^{\tau_{n,j}} \frac{1}{\rho!} \partial_{\tau}^{\rho} g(\tau_0, \sigma) d\sigma \\ &= -\frac{\varpi}{2} \sum_{k=1}^{\rho} \frac{(-1)^{k-1}}{\rho\,(\rho-k)!} \int_{\tau_{n,j-1}}^{\tau_{n,j}} \frac{e^{\tau_0 - \sigma}}{(-1)^k \left(-\tau_0 + \sigma\right)^k} d\sigma \\ &= \frac{\varpi}{2} \sum_{k=1}^{\rho} \frac{1}{\rho\,(\rho-k)!} \left[\Gamma\left(1-k, -\tau_0 + \tau_{n,j-1}\right) - \Gamma\left(1-k, -\tau_0 + \tau_{n,j}\right)\right]. \end{aligned}$$

2. For $\tau_{n,j-1} < \tau_0,\ \tau_{n,j} < \tau_0$:

$$\begin{aligned} B_{j\rho} &= \int_{\tau_{n,j-1}}^{\tau_{n,j}} \frac{1}{\rho!} \partial_{\tau}^{\rho} g(\tau_0, \sigma) d\sigma \\ &= -\frac{\varpi}{2} \sum_{k=1}^{\rho} \frac{(-1)^{\rho-1}}{\rho\,(\rho-k)!} \int_{\tau_{n,j-1}}^{\tau_{n,j}} \frac{e^{-(\tau_0 - \sigma)}}{\left(\tau_0 - \sigma\right)^k} d\sigma \\ &= -\frac{\varpi}{2} \sum_{k=1}^{\rho} \frac{(-1)^{\rho-1}}{\rho\,(\rho-k)!} \left[\Gamma\left(1-k, \tau_0 - \tau_{n,j}\right) - \Gamma\left(1-k, \tau_0 - \tau_{n,j-1}\right)\right]. \end{aligned}$$

It is important to remark that the entries satisfying $\tau_{n,j-1} < \tau_0$ and $\tau_{n,j} > \tau_0$ do not comply with the admissibility condition and are computed via (8). For these cases, the previous integral is divergent. □

2.2.3. Polynomial interpolation

As mentioned previously, an alternative to the Taylor series approach for obtaining a degenerate kernel is to use polynomial interpolation.

Let $(x_\rho)_{\rho=0,\dots,k}$ be a family of interpolation points in α (when dealing with β the procedure is analogous) and consider $(\mathcal{L}_\rho)_{\rho=0,\dots,k}$ to be the corresponding Lagrange polynomials that satisfy

$$\mathcal{L}_\rho(x_\theta) = \delta_{\rho,\theta}, \qquad \forall \theta \in \{0,\dots,k\}.$$

We interpolate the kernel and get its approximation in the form (10), for $f_\rho(\tau) = \mathcal{L}_\rho(\tau)$ and $h_\rho(\sigma) = g(x_\rho, \sigma)$.

It is necessary to find a set of interpolation points and corresponding Lagrange polynomials for each cluster. The former can be chosen to be the k-th order Chebyshev points, which, for an interval $[y_1, y_2]$, are given by

$$x_\rho := \frac{y_2 + y_1}{2} + \frac{y_2 - y_1}{2} \cos\left(\frac{2\rho+1}{2(k+1)}\pi\right).$$

The corresponding Lagrange polynomials can now be written as

$$\mathcal{L}_\rho(x) = \prod_{\theta=0,\theta\neq\rho}^{k} \frac{x - x_\theta}{x_\rho - x_\theta} \qquad \forall x \in [y_1, y_2].$$

To minimize the error when approximating the kernel g by its interpolant $\tilde{g}$, it turns out that the interpolation should be applied differently to the two arguments of the kernel [14], considering the diameter of the two subdomains:

$$\tilde{g}(\tau,\sigma) = \begin{cases} \sum_{\rho=0}^{k} g(x_\rho,\sigma)\,\mathcal{L}_\rho(\tau), & \text{if } \operatorname{diam}(\alpha) \leq \operatorname{diam}(\beta) \\ \sum_{\rho=0}^{k} g(\tau,x_\rho)\,\mathcal{L}_\rho(\sigma), & \text{otherwise} \end{cases}.$$

The admissibility condition (9) is implemented such that we take for the left hand side the $\min\{\operatorname{diam}(\alpha), \operatorname{diam}(\beta)\}$. As a consequence, the entries of matrices A and B from the factorization AB^T are computed using

$$A_{i\rho} = \frac{1}{h_{n,i}} \int_{\tau_{n,i-1}}^{\tau_{n,i}} \mathcal{L}_\rho(\tau)\, d\tau \qquad \text{and} \qquad B_{j\rho} = \int_{\tau_{n,j-1}}^{\tau_{n,j}} g(x_\rho,\sigma)\, d\sigma$$

if $\operatorname{diam}(\alpha) \leq \operatorname{diam}(\beta)$, and using

$$A_{i\rho} = \int_{\tau_{n,i-1}}^{\tau_{n,i}} g\left(\tau, x_\rho\right) d\tau \qquad \text{and} \qquad B_{j\rho} = \frac{1}{h_{n,j}} \int_{\tau_{n,j-1}}^{\tau_{n,j}} \mathcal{L}_\rho\left(\sigma\right) d\sigma$$

when $\mathrm{diam}(\beta) < \mathrm{diam}(\alpha)$.

Similarly to the previous subsection, these entries can be computed analytically. However, and to illustrate the flexibility of this approach, these integrals will be computed using numerical quadrature formulae for the results to be presented in section 5.

3. Eigenvalue Computations

This paper is concerned with the computation of a few eigenvalues and eigenvectors of matrix A_n, that is, to obtain a partial solution of (6), by means of iterative eigensolvers. In this section, we describe the methods very briefly, focusing on the required matrix operations that must be available in the implementation of $\mathcal{H}$-matrices.

3.1. Iterative Eigensolvers

There exist a large variety of iterative methods for the partial solution of eigenvalue problems. A detailed description can be found in [1]. Here we restrict our discussion to two families of methods, namely Krylov methods and Davidson methods.

Given the problem formulation

$$Ax = \lambda x, \tag{14}$$

where there are n eigenvalues λ and eigenvectors $x \neq 0$ that satisfy the equation, the goal is to find a subset of the eigenvalues in a given region of the spectrum (for the moment, we consider the simplest case where we seek the largest magnitude eigenvalues). Iterative eigensolvers are based on iteratively improving a subspace $\mathcal{V}$ in such a way that it eventually contains a good approximation of the eigenspace associated to the wanted eigenvalues.

Let $V \in \mathbb{R}^{n\times m}$, $m \ll n$, be a basis of $\mathcal{V}$ with $V^T V = I$, then the Rayleigh-Ritz projection method computes $H = V^T AV$ and uses its eigendecomposition $HY = Y\Theta$ to obtain approximate eigenpairs $(\theta_i, x_i = Vy_i)$ of A.

Krylov methods use the so-called Krylov subspaces associated with matrix A and a given initial vector v_1,

$$\mathcal{K}_m(A, v_1) = \mathrm{span}\{v_1, Av_1, A^2v_1, \ldots, A^{m-1}v_1\}\ , \tag{15}$$

where without loss of generality we assume that v_1 has unit length and is the first column of V.

The method of Arnoldi is an elegant algorithm that computes an orthonormal basis of the Krylov subspace and at the same time computes the projected matrix H, all this in an efficient and numerically stable way. In brief, the Arnoldi algorithm computes the m columns of V sequentially, where column v_{j+1} is the result of orthogonalizing Av_j with respect to previous columns, and normalizing. The orthogonalization is carried out by means of a Gram-Schmidt procedure, that removes all the components in the directions of $v_1, \ldots, v_j$. The computed quantities satisfy a relation of the form

$$AV_m = V_m H_m + \beta v_{m+1} e_m^T, \tag{16}$$

where H_m is an upper Hessenberg matrix, i.e., $h_{ij} = 0$ for $i > j+1$. The last term of the Arnoldi relation is the residual and gives an indication of how close is $\mathcal{K}_m(A, x_1)$ to an invariant subspace. In particular, β is used to assess the accuracy of the computed Ritz pairs. See [1] for additional details.

With Arnoldi, Ritz pairs will converge very fast provided that the initial vector v_1 is rich in the direction of the wanted eigenvectors. However, this is usually not the case and consequently many iterations would be required, but this cannot be allowed in a practical implementation in order to keep the storage requirements and the computational cost per iteration bounded. A workaround is to do a *restart* of the algorithm, that is, stop after m iterations and rerun the algorithm with a new v_1 computed from the recently obtained spectral approximations. An added benefit of this strategy is that it can be useful for driving convergence of the eigensolver towards a part of the spectrum different from the one targeted naturally by the method.

A very effective and elegant restart mechanism is the Krylov-Schur method [16]. It is defined by generalizing the Arnoldi decomposition (16) of order m to a Krylov decomposition of order m,

$$AV_m = V_m B_m + v_{m+1} b_{m+1}^T, \tag{17}$$

in which matrix B_m is not restricted to be upper Hessenberg and b_{m+1} is an arbitrary vector. Krylov decompositions are invariant under (orthogonal) similarity transformations, and this fact enables the truncation of the decomposition, compressing the subspace to a smaller dimension while keeping the relevant eigeninformation.

So far, the only operation required to matrix A is the matrix-vector product, which can be carried out very efficiently in the $\mathcal{H}$-matrix representation.

3.2. Computation of Eigenvalues around a Given Target

The Krylov-Schur method could in principle be used to compute any part of the spectrum, by keeping the wanted eigenvalues in the truncated factorization. Discarding the rest of the factorization has the effect of filtering out the information associated to the unwanted eigenvectors. However, when computing eigenvalues in the interior of the spectrum, this filter is not powerful enough, and components associated to extreme eigenvalues keep on appearing, thus hindering convergence to the wanted ones.

The simplest solution to compute eigenvalues closest to a given target, σ, is to use a spectral transformation, in such a way that eigenvalues are mapped to a different position while eigenvectors remain unchanged. One such transformation is the shift-and-invert technique, that solves the problem

$$(A - \sigma I)^{-1} x = \mu x, \tag{18}$$

where the transformed eigenvalues satisfy the simple relation $\mu = (\lambda - \sigma)^{-1}$. Eigenvalues λ closest to the target become dominant in the transformed spectrum, so Krylov methods will have a fast convergence. This can be implemented by simply replacing the action of A by that of $(A - \sigma I)^{-1}$ in the Krylov subspace expansion, that is, by solving linear systems with $(A - \sigma I)$ whenever a matrix-vector product is required. These linear systems must be solved very accurately, since Krylov methods can be very sensitive to numerical error introduced in the computation of the Krylov subspace, so in most applications a direct linear solver will be required, rather than an iterative method. It is generally claimed that the main drawback of the shift-and-invert technique is the high cost associated to direct linear solvers, since the memory requirements and computational effort can be very high for large, sparse matrices. In the case of the $\mathcal{H}$-matrix representation, this downside disappears because computing the factorization has much smaller cost, both in terms of storage and operations, as well as the corresponding triangular solves.

An alternative to the spectral transformation is the use of a preconditioned eigensolver such as Jacobi-Davidson. These methods expand the subspace in a different way, attempting to make the whole computation more robust with respect to numerical error in the application of the operator. This allows the use of iterative linear solvers such as GMRES in the so-called correction equation. Nevertheless, this does not seem to be the best approach in the context of $\mathcal{H}$-matrix representation, in view of the efficiency of matrix

factorization. From a practical perspective, to implement this kind of methods it is required to be able to build a preconditioner for matrix A, which can be as simple as its diagonal.

4. Implementation Details

In this paper, we illustrate the use of standard, freely available software tools for the assembly of numerical applications. In particular, we demonstrate a SLEPc-based code where the $\mathcal{H}$-matrix representation is implemented by plugging functionality from HLib. We describe these tools below. In a previous work [17], SLEPc was already used for solving the eigenproblem described in section 1, with the approach of explicitly storing the matrix elements.

SLEPc, the Scalable Library for Eigenvalue Problem Computations [2, 18], is a software package for the solution of large-scale eigenvalue problems on parallel computers. It can be used to solve standard and generalized eigenvalue problems, as well as other types of related problems such as the quadratic eigenvalue problem or the singular value decomposition. SLEPc can work with either real or complex arithmetic, in single or double precision, and it is not restricted to symmetric (Hermitian) problems. It can be used from code written in C, C++, and Fortran. In this work, we do not use the parallel capabilities of SLEPc.

SLEPc is built on top of PETSc (Portable, Extensible Toolkit for Scientific Computation, [19]), a parallel framework for the numerical solution of partial differential equations, and it uses primarily the basic data structures such as those for representing vectors and matrices.

SLEPc provides a collection of eigensolvers, most of which are based on the subspace projection paradigm. In particular, it includes a robust and efficient parallel implementation of Krylov-Schur method described in the previous section. Several Davidson-type solvers are included as well, in particular Generalized Davidson and Jacobi-Davidson, with various possibilities for the computation of the correction vector. In these solvers, the user can easily select which preconditioner to use. Apart from eigensolvers, some spectral transformations such as the shift-and-invert technique of (18) are available, where the user can compute interior eigenvalues with the aid of linear solvers and preconditioners included in PETSc.

The solvers in PETSc and SLEPc have a data-structure neutral implementation. This means that the computation can be done with different ma-

trix storage formats, and also even with a matrix that is not stored explicitly. By default, a matrix in PETSc is stored in a parallel compressed-row sparse format, where each processor stores a subset of rows. For implementing a matrix-free solver with so-called *shell* matrices, the application programmer has to create one of such matrices and define its operations, by binding a user-defined subroutine for each operation. Only the operations required by the actual computation need to be set, so in the simplest case it is sufficient to implement the matrix-vector product. For more advanced functionality, *e.g.*, preconditioning, other operations are required as well. We use this feature to interface our code to HLib.

HLib [6, 7] is a library for hierarchical matrices that was written by Lars Grasedyck and Steffen Börm. It is written in C and uses BLAS and LAPACK for lower-level algebraic operations. The library contains functions for $\mathcal{H}$- and $\mathcal{H}^2$-matrix arithmetics, the treatment of partial differential equations and a number of integral operators as well as support routines for the creation of cluster trees, visualization and numerical quadrature.

For building the $\mathcal{H}$-matrix we use HLib's *supermatrix* data structure, and recursively compute the cluster tree and populate it with admissible or inadmissible blocks, as described in section 2. We have also developed a straightforward parallel version of the $\mathcal{H}$-matrix generation, based on the OpenMP API for shared-memory parallel programming. In particular, we follow a *tasking* approach with the OpenMP `task` directive [20], where each recursive call constitutes a new task.

In our code, we have implemented three operations of the *shell* matrix: matrix-vector multiplication, shift of origin $A := A + \sigma I$, and extraction of the diagonal. These are simply calls to the corresponding HLib functions, appropriately wrapped according to PETSc convention. In addition, we have also implemented a *shell* spectral transformation in SLEPc that similarly implements a specialized version of the shift-and-invert technique of (18) by means of HLib's LU decomposition.

A final note about the implementation is that both the exponential-integral and incomplete Gamma functions are available via GSL, the GNU Scientific Library.

5. Numerical Experiments

In this section we show results for some numerical experiments that aim at illustrating the benefits of the $\mathcal{H}$-matrix representation with respect to con-

ventional storage, both in terms of performance and memory requirements.

The tests have been executed on a Linux workstation with an Intel Core i7 950 processor at 3,06 GHz with 8 MB of L3 cache memory and 8 GB of main memory. This processor has 4 cores with hyper-threading technology (a total of 8 virtual processors). The software configuration is based on Ubuntu Linux 10.04, with GCC 4.4.3 (the GNU C compiler, including support for OpenMP 3.0), PETSc 3.1, SLEPc 3.1, HLib 1.3, LAPACK 3.2.1, and GSL 1.13.

We present results when solving the transfer problem in stellar atmospheres described in section 1, equations (1)–(3). For all the tests, we chose to use a fixed value of the $\tau^\star$ parameter, in particular $\tau^\star = 4000$. We also set a constant value for the rank and minimum size of admissible blocks (`degree`=6 and `bound`=80, respectively), as well as $\eta = 1$ in (9).

In Table 1 we show the CPU time required for the matrix generation phase with dimension n varying from 4000 to 256000. With a uniform grid the resulting matrix is symmetric and the code takes this fact into consideration: the generation time for the symmetric case is almost half of the time required for the non-symmetric counterpart. The $\mathcal{H}$-matrix approach, either with Taylor or Lagrange approximations for computing the admissible blocks, represents a significant gain in generation time compared with the version with conventional sparse storage (note that in the sparse version we compute all matrix elements and then decide whether they are too small to be stored). The time reported for the SVD version includes the computation of matrix elements as in the sparse version as well as the time required for low rank approximation through SVD decomposition (with LAPACK). Although this variant is the most expensive one, if the problem is to be solved several times one may consider this approach since it allows both for a fixed rank-k and for a rank satisfying a prescribed tolerance, as mentioned in section 2. For large values of n the CPU time required to compute the entries is prohibitive for the sparse implementation. The slight differences reported for Taylor and Lagrange result from the fact that in the latter case we implemented numerical quadrature while for Taylor we used the formulae presented in section 2; the computation of the incomplete Gamma function at the required points is skewing the results a bit. For increasing problem size there is a constant growth factor less than three for these two approaches while the problem size is quadrupling. The growth factor respects the estimated $n \log(n)$ asymptotic cost, in contrast with the sparse version that follows n^2. Some values were not reported due to their high value.

Table 1: CPU time (in seconds) for the generation phase for $\tau^\star = 4000$ and varying n, with a uniform (left) and non-uniform (right) grid. The results correspond to `bound`=80 and `degree`=6.

	Uniform				Non-uniform			
n	Taylor	Lagrange	SVD	Sparse	Taylor	Lagrange	SVD	Sparse
4000	2.1	2.0	20.1	15.3	3.9	3.5	38.6	30.9
8000	6.1	5.7	99.6	61.8	10.4	9.4	182.7	123.3
16000	16.7	15.6	536.0	245.4	29.5	26.9	927.2	496.4
32000	49.6	47.0	3505.0	982.1	85.4	78.9	5630.7	1972.0
64000	140.6	133.5	–	–	245.4	227.2	–	–
128000	394.4	374.4	–	–	690.3	637.9	–	–
256000	1019.8	958.3	–	–	1783.4	1616.1	–	–

Table 2: Number of stored elements in the case of dense, sparse and $\mathcal{H}$-matrix representation, for the non-uniform grid case in Table 1.

n	Dense	Sparse	$\mathcal{H}$-matrix	Compression
4000	$1.6 \cdot 10^7$	$2.9 \cdot 10^5$	$1.9 \cdot 10^6$	12.0%
8000	$6.4 \cdot 10^7$	$1.2 \cdot 10^6$	$4.2 \cdot 10^6$	6.5%
16000	$2.6 \cdot 10^8$	$4.5 \cdot 10^6$	$9.1 \cdot 10^6$	3.5%
32000	$1.0 \cdot 10^9$	$1.8 \cdot 10^7$	$1.9 \cdot 10^7$	1.9%
64000	$4.1 \cdot 10^9$	–	$4.1 \cdot 10^7$	1.0%
128000	$1.6 \cdot 10^{10}$	–	$8.6 \cdot 10^7$	0.5%
256000	$6.6 \cdot 10^{10}$	–	$1.9 \cdot 10^8$	0.28%

Table 2 complements the previous comments, showing the number of stored elements for all approaches. Note that the actual memory requirements for sparse storage are quite large, since the space needed for indices is considerable, while for the $\mathcal{H}$-matrix representation the overhead in negligible. The last column shows the compression factor for $\mathcal{H}$-matrix format as a percentage of the full (dense) storage, revealing noteworthy gains for increasing values of n.

Table 3 reports on the CPU time for the solution phase using Taylor, Lagrange and SVD data-sparse representation as well as the sparse approach. Since the spectrum is tightly clustered (see Table 4 for the five largest eigenvalues with relative tolerance on the residual of 10^{-7}), the shift-and-invert technique is required to enable convergence of the Krylov-Schur method. In the following, an LU factorization on the $\mathcal{H}$-matrix representation is used in

Table 3: CPU time (in seconds) for the solution phase for $\tau^\star = 4000$ and varying n, with a non-uniform grid (non-symmetric case). The results correspond to `bound`=80 and `degree`=6. Times reported for the factorization are included in those for the solution.

	Solution				Factorization	
n	Taylor	Lagrange	SVD	Sparse	Taylor	Lagrange
4000	0.3	0.3	0.3	0.2	0.3	0.3
8000	0.5	0.5	0.5	1.1	0.4	0.4
16000	1.0	1.1	1.2	8.2	0.8	0.9
32000	2.6	2.7	2.9	66.3	2.1	2.3
64000	6.5	6.8	–	–	5.6	5.9
128000	16.4	17.3	–	–	14.5	15.4
256000	47.5	49.3	–	–	39.3	41.3

the linear solver required in the application of the shift-and-invert operator. The factorization is the most costly operation but is performed only once, while triangular solves are required at each iteration of the eigensolver. In the table we show the factorization time as well as the total solution time. For these tests, we used a Krylov basis of dimension 16, and with this size the method does not need to restart (except for the matrix of $n = 256000$ where 2 restarts are required).

As expected, the computation of eigenpairs with the implementations of the data-sparse representation is very fast compared to the sparse storage, which shows a fast degradation in performance for increasing dimension. The SVD approach is competitive with Lagrange and Taylor approximations, and results for SVD on large values of n are not reported only due to the high generation time. As mentioned above, the present problem is hard to solve since for increasing values of n, and for fixed $\tau^\star$, the eigenvalues tend to become more and more clustered. For problems with better separation of the spectrum, the shift-and-invert step can be avoided and consequently its computational cost.

As mentioned at the end of section 3, Davidson methods do not seem too appropriate in the context of hierarchical matrices, since the shift-and-invert technique is very cheap in this case. However, we wanted to do some experiments. With Jacobi (diagonal) preconditioning, we were able to solve the problem (although after many iterations) by tuning the parameters of SLEPc's Davidson solver. For instance, for $n = 8000$ with uniform grid, the response time is 15.5 seconds, as opposed to 0.5 seconds with shift-and-invert

Table 4: Computed eigenvalues for the case of a uniform grid with $n = 16000$ and $\tau^\star = 4000$.

Eigenvalue	Taylor	Lagrange	SVD
λ_1	0.749999843422	0.749999843459	0.749999843598
λ_2	0.749999374216	0.749999374253	0.749999374391
λ_3	0.749998592208	0.749998592245	0.749998592383
λ_4	0.749997497401	0.749997497438	0.749997497576
λ_5	0.749996089801	0.749996089838	0.749996089976

Table 5: Execution time (in seconds) for the matrix generation in parallel (for different number of threads, p) corresponding to the two longest times in Table 1 (non-uniform grid, Taylor with $n = 256000$ and SVD with $n = 32000$).

	Taylor 256000		SVD 32000	
p	Time	Speedup	Time	Speedup
1	1783.4	–	5630.7	–
2	891.9	1.99	3097.2	1.82
4	446.0	3.99	1983.4	2.83
6	381.0	4.68	1875.1	3.00
8	332.9	5.35	1834.0	3.07

Krylov-Schur. A much powerful preconditioner is to use the LU factorization, but then the behaviour is almost identical to shift-and-invert. Again, we remark that in other applications with a different spectrum, Davidson solvers could be more useful than in this case.

Regarding parallelization of the generation phase, the multi-threaded version was analyzed up to 8 threads. Table 5 shows the measured execution times along with the achieved speedups, for the two longest times in Table 1 (non-uniform grid, Taylor with $n = 256000$ and SVD with $n = 32000$). In the case of the Taylor approximation, speedup is virtually ideal up to 4 threads and decays significantly later. This can be attributed in part to the fact that only 4 physical cores are available. In the case of SVD generation, speedup is much worse, thus revealing a problem with load imbalance, due to the fact that the high cost of the decomposition (cubic in the matrix block size) makes parallel tasks differ wildly in duration.

6. Conclusion

In this paper, we use iterative eigensolvers to compute a few eigenelements of an integral operator appearing in a radiative transfer equation in stellar atmospheres. The computation is intensive both in time and memory requirements, since large dimensional cases are to be treated. The generation of the matrix is expensive since it requires multiple evaluations of the exponential-integral function, and the solution phase is costly as well, since the shift-and-invert technique is necessary due to the clustering of the eigenvalues. The $\mathcal{H}$-matrix representation provided by HLib was integrated in the SLEPc and PETSc frameworks to tackle these two difficulties. We report on the numerical low-rank approximations developed, on the details of the integration of the libraries under consideration, and present a brief explanation of the application problem and its most interesting characteristics. The combined use of efficient numerical methods and the clever data storage provides a fast answer, thus enabling the solution of large dimensional problems. Numerical tests illustrate the success of the proposed solutions.

We have addressed the issue of parallelization only partially, with a straightforward approach. It remains as a future investigation the extension of the parallelization to the solution stage as well. Another possible extension is the use of the more memory-efficient $\mathcal{H}^2$-matrix format [21].

Acknowledgements. We would like to express our gratitude to Mario Ahues and Steffen Börm for their valuable comments and remarks.

This work was partially supported by the Spanish Ministerio de Ciencia e Innovación under projects TIN2009-07519 and AIC10-D-000600 and by Fundação para a Ciência e a Tecnologia - FCT under project FCT/MICINN proc 441.00.

References

[1] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, H. van der Vorst (Eds.), Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2000.

[2] V. Hernandez, J. E. Roman, V. Vidal, SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems, ACM Transactions on Mathematical Software 31 (3) (2005) 351–362.

[3] W. Hackbusch, A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices, Computing 62 (2) (1999) 89–108.

[4] W. Hackbusch, B. N. Khoromskij, A sparse $\mathcal{H}$-matrix arithmetic. Part II: application to multi-dimensional problems, Computing 64 (1) (2000) 21–47.

[5] M. Lintner, The eigenvalue problem for the 2D Laplacian in $\mathcal{H}$-matrix arithmetic and application to the heat and wave equation, Computing 72 (3) (2004) 293–323.

[6] S. Börm, L. Grasedyck, W. Hackbusch, Introduction to hierarchical matrices with applications, Engineering Analysis with Boundary Elements 27 (5) (2003) 405–422.

[7] S. Börm, L. Grasedyck, W. Hackbusch, Hierarchical Matrices, Lecture Note 21 of Max-Planck-Institut für Mathematik in den Naturwissenschaften, Leipzig (2003).

[8] B. Rutily, L. Chevallier, The finite Laplace transform for solving a weakly singular integral equation occurring in transfer theory, Journal of Integral Equations and Applications 16 (4) (2004) 389–409.

[9] M. Ahues, F. D. d'Almeida, A. Largillier, O. Titaud, P. Vasconcelos, An L^1 refined projection approximate solution of the radiation transfer equation in stellar atmospheres, Journal of Computational and Applied Mathematics 140 (1-2) (2002) 13–26.

[10] M. Abramowitz, I. A. Stegun (Eds.), Handbook of Mathematical Functions, Dover, New York, USA, 1965.

[11] M. Ahues, F. D. d'Almeida, A. Largillier, P. B. Vasconcelos, Defect correction for spectral computations for a singular integral operator, Communications on Pure and Applied Analysis 5 (2) (2006) 241–250.

[12] F. d'Almeida, O. Titaud, P. B. Vasconcelos, A numerical study of iterative refinement schemes for weakly singular integral equations, Applied Mathematics Letters 18 (5) (2005) 571–576.

[13] O. Titaud, Reduction of computation in the numerical resolution of a second kind weakly singular Fredholm equation, in: C. Constanda,

M. Ahues, A. Largillier (Eds.), Integral Methods in Science and Engineering: Analytic and Numerical Techniques, Birkhäuser, 2004, pp. 255–260.

[14] S. Börm, L. Grasedyck, Low-rank approximation of integral operators by interpolation, Computing 72 (3) (2004) 325–332.

[15] M. Bebendorf, Hierarchical Matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems, Vol. 63 of Lecture Notes in Computational Science and Engineering, Springer-Verlag, 2008.

[16] G. W. Stewart, A Krylov–Schur algorithm for large eigenproblems, SIAM Journal on Matrix Analysis and Applications 23 (3) (2001) 601–614.

[17] P. B. Vasconcelos, O. Marques, J. E. Román, High-performance computing for spectral approximations, in: C. Constanda, M. E. Pérez (Eds.), Integral Methods in Science and Engineering, Volume 2: Computational Methods - IMSE 2008, Birkhäuser, 2010, pp. 351–360.

[18] V. Hernandez, J. E. Roman, A. Tomas, V. Vidal, SLEPc users manual, Tech. Rep. DSIC-II/24/02 - Revision 3.1, D. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia (2010).

[19] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, B. F. Smith, H. Zhang, PETSc users manual, Tech. Rep. ANL-95/11 - Revision 3.1, Argonne National Laboratory (2010).

[20] E. Ayguadé, N. Copty, A. Duran, J. Hoeflinger, Y. Lin, F. Massaioli, X. Teruel, P. Unnikrishnan, G. Zhang, The design of OpenMP tasks, IEEE Transactions on Parallel and Distributed Systems 20 (3) (2009) 404–418.

[21] S. Börm, Construction of data-sparse $\mathcal{H}^2$-matrices by hierarchical compression, SIAM Journal on Scientific Computing 31 (3) (2009) 1820–1839.