CHAPTER 7

Iterative schemes for the solution of systems of equations arising from the DRM in multidomains

M.I. Portapila\textsuperscript{1} & H. Power\textsuperscript{2}
\textsuperscript{1}Wessex Institute of Technology, UK.
\textsuperscript{2}Department of Mechanical Engineering, University of Nottingham, UK.

Abstract

The aim of this work is to carry out a systematic experimental study of the use of iterative techniques in the context of solving the linear system of equations arising from the solution of the diffusion–convection equation with variable velocity field through the use of the dual reciprocity method in multidomains (DRM-MD). We analyse the efficiency and accuracy of the computed solutions obtained from the DRM-MD integral equation numerical approach applying various iterative algorithms. For every iterative method tested, we consider a set of different preconditioners, depending on the features of the input matrix to be solved with the chosen method. To check the accuracy of the solutions obtained through the selected iterative methods, they are contrasted against the solutions obtained applying some direct methods such as singular value decomposition, Golub’s method and Cholesky decomposition. The numerical results are also compared with a benchmark analytical solution. Furthermore, we present a comparative analysis of the linear systems of algebraic equations obtained from DRM-MD considering two approximating functions: the conical function $r$ plus a constant, i.e. $(1 + r)$, and the augmented thin plate spline, both of them radial basis functions.

1 Introduction

Intensive research has taken place on the behaviour of the system of equations arising from boundary element techniques and its solution applying iterative methods.
The matrices of coefficients originated from standard single-zone boundary element method (BEM) are unsymmetric, fully populated and very large. This fact represents a considerable disadvantage when comparing with the classical domain methods, such as the finite element method (FEM) or finite difference method, since more computer operations are involved.

To overcome this drawback when dealing with the BEM for large problems, it is usual to use the method of domain decomposition, in which the original domain is divided into subregions, and on each of them the full integral representation formulae are applied. At the interfaces of the adjacent subregions, the corresponding full matching conditions are imposed. The domain decomposition approach is itself a very powerful and popular scheme in numerical analysis that has recently increased its popularity due to its use in parallel computing algorithms (for more details see [1]). While the BEM matrices, which arise in the single domain formulation, are fully populated, the subregion formulation leads to block-banded matrix systems with one block for each subregion and overlaps between blocks when subregions have a common interface. When using continuous elements of high orders (more than constant), the application of the matching conditions at common interfaces, i.e. the matrix assembly, leads to an overdetermined system of algebraic equations (as more subregions are defined the bigger the overdetermination). Several schemes are known that reduce the overdetermined system to a closed system (for more details see [2]). The simplest possible scheme is obtained by expressing the derivatives of the field variables at the common nodes between more than two subregions in terms of the variables themselves by using the interpolation functions or by a finite difference approximation. A way of avoiding this problem is by using discontinuous elements in such common nodes, obtaining in this way a closed matrix system at the expense of having a larger number of unknown variables. Alternatively, one can solve the complete overdetermined system. In this regard, several algorithms are known in the mathematical literature which can numerically solve this kind of system, the most popular being the singular value decomposition (SVD) and the least-square approaches.

In the limit of a very large number of subregions, the resulting internal mesh pattern looks like a finite element grid. Kane et al. [3], and Guru Prasad et al. [4] report that multi-zone problems use considerably less storage that single-zone applications. They noted that the performance of the conjugate gradient (CG) iterative method for the normal equation approach was improved by multi-zone modelling, but it did not improve the performance of the GMRES approach. In all the studied cases the converged solutions were as accurate as those derived using the direct equation. They remarked that for both single-zone and multi-zone problems, the preconditioned iterative approaches were generally faster than the direct methods.

Further increase in the number of applications of the BEM has been hampered by the need to operate with relatively complex fundamental solutions or by the difficulties encountered when these solutions cannot be expressed in a closed form, and also when the technique is applied to nonlinear and time-dependent problems. In the BEM for this kind of problems, it is common to use an integral representation formula based on a partial differential equation for which a closed form
expression of the fundamental solution is known, and express the remaining terms
of the original equation as domain integrals. In the early BEM analysis the evalua-
tion of domain integrals was done using cell integration, a technique, which while
effective and general, made the method lose its boundary-only nature, which is one
of its attractive features, introducing additional internal discretization. Although
good results can be obtained using the cell integration technique, this approach,
for certain applications, is several orders of magnitude more time consuming than
domain methods. This computational cost mainly depends on the fact that the solu-
tion at each surface or internal point must involve the evaluation of the complete
surface integrals.

An implementation of the subregion BEM formulation in this limiting case, i.e.
with a very large number of subregions, including cell integration at each subregion
has been called by Taigbenu and collaborators the Green element method (GEM)
(see [5, 6]), in which a finite difference approximation is used to reduce the over-
determined system of equations. In these conditions, the resultant coefficient matrix
is as sparse as that encountered in FEM and, therefore, its solution is as efficient as
in the domain approach, and the results are as accurate as those of the BEM.

Using a idea similar to the GEM, Ramsak and Skerget [7] solved viscous fluid
flow problems governed by the complete Navier–Stokes equation in terms of a
mixed boundary-domain integral approach and a subdomain technique. In this way
they reach a final discretized matrix system that is overdetermined, sparse and block
banded, which they solve using the iterative linear least squares solver (LSQR) pro-
posed by Paige and Saunders [8], accelerating the convergence through a diagonal
preconditioning.

Several methods have been developed to take domain integrals to the boundary
in order to eliminate the need for internal cells (boundary-only BEM formulations).
One of the most popular to date is the dual reciprocity method (DRM) introduced by
Nardini and Brebbia [9]. This method is closely related to the particular integrals
 technique (PIT) introduced by Ahmad and Banerjee [10], which is also used to
 transform domain integrals into boundary integrals. In the latter method a particular
 solution satisfying the non-homogeneous partial differential equation (PDE) is first
 found and then the remaining part of the solution satisfying the corresponding
 homogeneous PDE is obtained by solving the corresponding integral equations.
The boundary conditions for the homogeneous PDE must be adjusted to ensure
that the total solution satisfies the boundary conditions of the original problem.
The DRM also uses the concept of particular solutions, but instead of solving
for the particular solution and the homogeneous solution separately, it applies the
divergence theorem to the domain integral terms and converts the domain integrals
into equivalent boundary integrals (for more information on some other techniques
for the evaluation of domain integrals see [11]).

A major problem encountered with the PIT and the DRM is that the resulting
algebraic system consists of a series of matrix multiplications of fully populated
matrices. When only few internal points are required in the PIT or the DRM,
the resulting computing time is in general smaller than the one required by the
cell integration scheme, but is still costly in comparison with domain approaches.
Besides, in complex problems these two approaches have been limited only to small values of the nonlinear terms, or small Peclet numbers in the case of the convection–diffusion equation. From these limitations, it appears that these boundary-only formulations cannot be extended to solve highly domain-dominant problems.

As in the case of the standard BEM technique, the algebraic system of equations resulting from the DRM in a single domain are large, fully populated and unsymmetric. Bulgakov et al. [12] showed that the matrices of these systems have properties that allow the efficient use of Krylov iterative solvers such as CG squared and generalized minimal residual (GMRES) methods with Jacobi preconditioning. The GMRES method appeared to be more efficient than the CG squared method.

Popov and Power [13] found that the DRM approach can be substantially improved by using domain decomposition. The idea of using domain decomposition to improve the accuracy of the DRM approach was inspired by the work of Kansa and Carlson [14] on the radial basis function (RBF) data approximations, where they observed that the best approximation is obtained when the original domain is split into matching subdomains.

The aim of this work is to carry out a systematic experimental study of the use of iterative techniques in the context of solving the linear system of equations arising from the solution of the diffusion–convection equation with variable velocity field through the use of the dual reciprocity method in multidomains (DRM-MD). Besides, considering different iterative solutions, we present a comparative analysis of the performance of two different RBFs used in the interpolation of the DRM approach, i.e. the conical function plus a constant \((1 + r)\) and the augmented thin-plate spline (ATPS).

### 2 Preliminary remarks on the BEM

The BEM is a numerical approach that involves the transformation of original boundary integral equations into an equivalent algebraic form suitable to be solved numerically.

This approach consists of the discretization of the boundary \(\Gamma\) into a series of elements, this procedure being what gives the name to the method. The boundary integrals then are evaluated numerically over these segments or elements in which the boundary is divided. The BEM, therefore, consists of the following steps

1. The boundary \(\Gamma\) is discretized into a series of elements over which the potential and its normal derivative are assumed to vary according to interpolation functions. The geometry of the elements can be represented by constant, linear, quadratic or even higher-order elements, depending on how accurately we may want to represent the geometry of the domain.
2. The discretized integral equation is applied to a number of nodes within each element (collocation technique), where the values of the potential and its normal derivative are associated.
3. The integrals over each boundary element are carried out by using an adequate numerical quadrature.
4. By imposing the prescribed boundary conditions of the problem, a system of linear algebraic equations is obtained, which can be solved either using direct or iteratives methods. The solution of this system of equations will produce the remaining boundary data. It is interesting to note that the unknowns are a mixture of the potential and its normal derivative, rather than only the potential as it would be the case in finite elements.

5. At last, if values of the function $u$ and its derivative are required at internal points, it is possible to calculate them taking into account that the fundamental solution will be acting inside the domain and all values $u$ and the derivatives on the boundary, present on the new system to be solved are already known.

2.1 The BEM and the domain integrals

Domain integrals in the BEM may arise as a consequence of different situations, such as sources inside the domain, initial states, nonlinear terms, etc. When we have a Poisson-type equation ($\nabla^2 u = b$), the integral equation (1) has a domain integral term which needs to be performed.

Different methods are available to compute the domain integrals. A simple way of computing the domain term

$$c(\xi)u(\xi) + \int_{\Gamma} u(y) \frac{\partial u^*(\xi, y)}{\partial n} d\Gamma + \int_{\Omega} b(y)u^*(\xi, y) d\Omega = \int_{\Gamma} u^*(\xi, y) \frac{\partial u(y)}{\partial n} d\Gamma$$

is by subdividing the domain $\Omega$ into a series of internal cells over which a numerical integration formula can be applied. By doing so, for each boundary point $\xi$ we can write the domain integral as

$$d_\xi = \iint_{\Omega} bu^* d\Omega = \sum_{N_e} \left[ \sum_{k=1}^{K} w_k (bu^*_k) \right] \Omega_e$$

where $w_k$ are the numerical integration weights, the function $(bu^*)$ must be evaluated at the $K$ integration points on each cell, $N_e$ is the number of cells into which $\Omega$ has been divided, $\Omega_e$ is the area of cell $e$, and $K$ is the number of integration points on each cell. The term $d_\xi$ is the result of the numerical integration and we calculate it for each position of the $N$ boundary nodes. The whole set of equations for the $N$ nodes in matrix form can be expressed as

$$d + Hu = Gq$$

Another way of integrating the domain terms consists of using the Monte Carlo technique [15, 16] based on the integration over a system of random points rather than applying a regular integration grid as proposed in the cell integration method.

We shall also mention an alternative way of including internal sources into the formulation for cases when the function $b$ is harmonic in $\Omega$ [17], by transforming the corresponding domain integral into equivalent boundary integrals. This is done...
by means of a new function \( w^* \) such that \( \nabla^2 w^* = u^* \), and through the Green’s second identity the domain integral is reduced to

\[
\int \int_{\Omega} b u^* \, d\Omega = \int \int_{\Omega} (b \nabla^2 w^* - w^* \nabla^2 b) \, d\Omega = \oint_{\Gamma} \left( b \frac{\partial w^*}{\partial n} - w^* \frac{\partial b}{\partial n} \right) \, d\Gamma
\]

(4)

the function \( w^* \) is given in [18] as

\[
w^* = \frac{r^2}{8 \pi} \left[ \ln \left( \frac{1}{r} \right) + 1 \right]
\]

(5)

The multiple reciprocity method [19] is another technique for transforming domain integrals to the boundary which can be seen as a generalization of the just mentioned approach. It makes use of a set of higher-order fundamental solutions in order to apply the Green’s second identity to each term of the sequence. As a result the method can lead in the limit to the exact boundary-only formulation of the problem.

Finally we may consider the use of particular solutions as another way of solving eqn (1) without having to compute any domain integrals. This can be achieved by splitting the function \( u \) into a particular solution and the solution of the associated homogeneous equation. The main disadvantage of this methodology is that particular solutions are in many cases difficult to obtain when not impossible.

The dual reciprocity approximation introduced by Nardini and Brebbia [9] is a generalization of the use of particular solutions where the basic idea is to expand the non-homogeneous terms as a series of known interpolating functions. This method is briefly described in the following section.

3 The dual reciprocity approximation

The dual reciprocity approximation, first proposed in 1982 [9] can be summarized as a technique that enables a ‘boundary only’ solution to problems arising from the BEM with domain integrals. Basing its most remarkable feature on the fact that it does not depend on obtaining a new particular solution for each case under consideration.

We consider here, for simplicity, the mathematical formulation of the DRM for the Poisson equation, extending this methodology later on to the non-transient diffusion–convection equation, for which we will be presenting numerical analyses.

Hence, we consider

\[
\nabla^2 u = b
\]

(6)

where \( b \) is a known function of position. The solution to eqn (6) can be expressed as the sum of the solution of the homogeneous part plus a particular solution \( \hat{u} \), such that

\[
\nabla^2 \hat{u} = b
\]

(7)
Since a particular solution $\hat{u}$ is not easy to be found, the DRM proposes an approximation for $b$ in terms of a series of particular solutions $\hat{u}_j$. The number of $\hat{u}_j$ will be as many as the nodes we will prescribe to study the problem. Therefore, the approximation will take place at $N$ boundary nodes (the same as in the BEM) along with $L$ internal nodes, where the term $b$ is interpolated as follows

$$b \simeq \sum_{j=1}^{N+L} \alpha_j f_j$$

(8)

Here the $\alpha_j$ are unknown coefficients and the $f_j$ are approximating functions. The particular solutions $\hat{u}_j$ and the approximating functions $f_j$ are related through the Laplacian operator

$$\nabla^2 \hat{u}_j = f_j$$

(9)

and there will be $N + L$ values of $\hat{u}_j$ (Fig. 1).

The functions $f_j$ are geometry dependent, and when we substitute eqn (9) into (8) it yields

$$b = \sum_{j=1}^{N+L} \alpha_j (\nabla^2 \hat{u}_j)$$

(10)

Combining eqns (6) and (10) we find

$$\nabla^2 u = \sum_{j=1}^{N+L} \alpha_j (\nabla^2 \hat{u}_j)$$

(11)
Applying the Green’s second identity to eqn (11) and using the Laplacian fundamental solution \( u^* \) as the auxiliary potential, the following surface-only integral representation formula, in terms of the interpolation coefficients \( \alpha_j \), is obtained

\[
cu - \int_G u^* \left( \frac{\partial u}{\partial n} \right) d\Gamma + \int_G q^* u d\Gamma = \sum_{j=1}^{N+L} \left\{ \alpha_j \left( \int_G \hat{q}_j u^* d\Gamma - \int_G \hat{u}_j q^* d\Gamma + c \hat{u}_j \right) \right\}
\]

(12)

where the coefficient \( c \), the fundamental solution \( u^* \) and the normal derivative \( q^* \) are those defined in the standard BEM technique, and \( \hat{q}_j = \frac{\partial \hat{u}_j}{\partial n} \).

In eqn (1) Green’s second identity was applied only to the left-hand side of the expression. Now this operation (weighted residual technique, Green’s identity or a reciprocity principle) is applied to both sides of the equation in order to take all terms to the boundary. This procedure gives its name to the DRM.

The discretized form of eqn (12) is

\[
c_i u_i + \sum_{k=1}^{N} H_{ik} u_k - \sum_{k=1}^{N} G_{ik} q_k = \sum_{j=1}^{N+L} \left\{ \alpha_j \left( \sum_{k=1}^{N} H_{jk} \hat{u}_k - \sum_{k=1}^{N} G_{jk} \hat{q}_k + c_i \hat{u}_j \right) \right\}
\]

(13)

where \( H_{ik} \) and \( G_{ik} \) are the standard influence coefficients coming from the integrations of \( u^* \) and \( q^* \) at each boundary element and the functions \( \hat{u} \) and \( \hat{q} \) for a given interpolant function \( f \) can be found analytically. In this work, the linear boundary element is used in the discretization of the above integral equation.

After collocating in all boundary nodes \( i \), we can write eqn (13) in matrix notation, i.e.

\[
Hu - Gq = \sum_{j=1}^{N+L} \alpha_j (H\hat{u}_j - G\hat{q}_j)
\]

(14)

where the terms \( c_i \) have been included in the diagonal of \( H \). Furthermore, considering \( \hat{u}_j \) and \( \hat{q}_j \) as columns of the matrices \( \hat{U} \) and \( \hat{Q} \) we can rewrite eqn (14) as

\[
Hu - Gq = (H\hat{U} - G\hat{Q})\alpha
\]

(15)

In the above equation the coefficients \( \alpha \) are determined by the inversion of the interpolation matrix \( F \), i.e.

\[
\alpha = F^{-1} b
\]

(16)

where \( F \) is the matrix resulting from the above interpolation with \( f \) as the approximating function.

After applying the boundary conditions to expression (15) we will be able to get a boundary solution to problems governed by a Poisson equation. To obtain the solution at internal nodes we calculate \( u_i \) from eqn (13), recalling that in this case \( c_i = 1 \), and this procedure will involve the multiplication of known vectors and matrices.
3.1 Analysis of the convection–diffusion equation using DRM

In this section we will focus on a boundary-only integral method for the solution of non-transient convection–diffusion problems. The general form of the convection–diffusion equation can be written as

\[ D_x \frac{\partial^2 u}{\partial x^2} + D_y \frac{\partial^2 u}{\partial y^2} - v_x \frac{\partial u}{\partial x} - v_y \frac{\partial u}{\partial y} - ku = 0 \]  

(17)

where \( u \) is a concentration of a substance, a temperature, etc., \( D_x \) and \( D_y \) are dispersion coefficients, \( v_x \) and \( v_y \) are velocities and \( k \) is a decay parameter. When \( v_x = v_y = k = 0 \) then eqn (17) reduces to the diffusion equation commonly used in heat transfer problems. If \( D_x = D_y \) the problem is isotropic, if not it is orthotropic.

Thus far, we have considered the DRM for handling Poisson equations where the right-hand side of eqn (6) is a known function of position, namely \( \nabla^2 u = b(x, y) \).

In such applications the solution is divided into a boundary solution and an internal solution.

The use of the DRM for solving convection–diffusion problems by means of the fundamental solution available for the Laplace equation requires that all terms in eqn (17) not including the Laplacian, are transferred to the right-hand side as forcing terms, to obtain an equation of the type

\[ \nabla^2 u = b(x, y, u) \]  

(18)

In the isotropic case, where \( D_x = D_y = D \), the non-homogeneous term \( b \) is then given by

\[ b = \frac{1}{D} \left( v_x \frac{\partial u}{\partial x} + v_y \frac{\partial u}{\partial y} + ku \right) \]  

(19)

For this class of problems it is no longer possible to separate boundary and interior solutions, resulting in a coupled system of equations in which both sets of values are calculated at once.

3.1.1 The decay term

This term is

\[ b_1 = \frac{k}{D} u \]  

(20)

thus from eqn (16) we have in matrix notation

\[ \mathbf{b}_1 = \mathbf{a F} \]

\[ \mathbf{a} = \mathbf{F}^{-1} \mathbf{b}_1 = \mathbf{F}^{-1} \frac{k}{D} \mathbf{u} \]  

(21)

and we are able to recast eqn (15) as follows

\[ \mathbf{H u} - \mathbf{G q} = (\mathbf{H} \hat{\mathbf{U}} - \mathbf{G} \hat{\mathbf{Q}}) \mathbf{F}^{-1} \frac{k}{D} \mathbf{u} \]  

(22)
Since $\alpha$ cannot be obtained explicitly, as it was the case for Poisson-like equations, expression (21) will always be necessary. Matrix $F$ depends only on the geometry of the problem and has no relation to either governing equations or boundary conditions. In practice, it can be computed once and stored to save time and computing resources, in order to be ready to use in subsequent analyses involving the same discretization.

If we define

$$S = (H\hat{U} - G\hat{Q})F^{-1}$$

(23)

which calculation involves only known matrices, we will be able to rewrite eqn (22) as follows

$$Hu - Gq = \frac{k}{D}Su$$

(24)

It is important to note that this equation involves $N$ values of $u$ and $q$ on the boundary that are unknown, along with $L$ values of $u$ at interior nodes also unknown. While $q$ is not defined for internal nodes.

3.1.2 The convective term

First-order space derivatives are rather frequent in engineering problems, and adapting them into the DRM treatment allows us to recover the boundary-only nature of the BEM.

This term is of the form

$$b_2 = \frac{1}{D} \left( v_x \frac{\partial u}{\partial x} + v_y \frac{\partial u}{\partial y} \right)$$

(25)

To complete the DRM formulation of this term an algorithm must be established to relate the values of $b$ to the nodal values of the concentration $u$. This is usually done by expressing the concentration at a domain point in terms of the approximation functions $f_i$, i.e.

$$u = F\beta; \quad (i = 1, \ldots, N + L)$$

(26)

where $\alpha \neq \beta$.

Differentiation of eqn (26) produces:

$$\frac{\partial u}{\partial x_l} = \frac{\partial F}{\partial x_l} \beta \quad l = 1, 2$$

(27)

Rewriting eqn (26) as $\beta = (F)^{-1}u$, eqn (27) becomes:

$$\frac{\partial u}{\partial x_l} = \frac{\partial F}{\partial x_l} (F)^{-1}u \quad l = 1, 2$$

(28)

Equation (28) states the fact that the nodal values of the derivatives are in terms of the product of two known matrices which depend only on the geometry and the nodal values of $u$. 
Returning to eqn (15) and replacing $\partial u/\partial x_l$ from expression (28) into (16) we arrive to

$$Hu - Gq = (\hat{H}U - \hat{G}Q) \frac{F^{-1}}{D} \left( v_x \frac{\partial F}{\partial x} F^{-1} + v_y \frac{\partial F}{\partial y} F^{-1} \right) u$$

or using expression (23)

$$Hu - Gq = S \frac{1}{D} \left( v_x \frac{\partial F}{\partial x} F^{-1} + v_y \frac{\partial F}{\partial y} F^{-1} \right) u$$

Finally, the complete DRM equation for the non-transient convection–diffusion problem, with expression (19) as forcing term, results in a matrix system for the normal derivative of the concentration at the surface points, and for the concentration at the surface and internal points, in terms of only the surface integral of the influence coefficients $H$ and $G$, i.e.

$$Hu - Gq = (\hat{H}U - \hat{G}Q) \frac{F^{-1}}{D} \left( v_x \frac{\partial F}{\partial x} F^{-1} + v_y \frac{\partial F}{\partial y} F^{-1} + k \right) u$$

or considering eqn (23)

$$Hu - Gq = S \frac{1}{D} \left( v_x \frac{\partial F}{\partial x} F^{-1} + v_y \frac{\partial F}{\partial y} F^{-1} + k \right) u$$

### 3.2 Overview of DRM in multidomains

This treatment of the domain integral (eqn (31)), allows us to handle a boundary-only integral formulation of this type of problems. However, to improve the accuracy of the solution we need to increase the number of internal nodes in the domain. As the number of internal nodes increases, the multiplication of the above set of fully populated matrices requires higher computational effort.

A technique to prevent this drawback is to subdivide the domain into subregions where the DRM can be applied one by one. This multiple region technique is named the dual reciprocity method in multidomains (DRM-MD), and has proved to possess better convergence and better approximation properties than the application of the DRM directly to the single domain [20, 21].

The subdomains will have common interfaces with adjacent subdomains where compatibility conditions, also called matching conditions, have to comply. At each of these interfaces, the flux leaving one subregion has to be equal to the flux entering the other. Therefore, it is necessary that the following flux-matching conditions hold at the $m$th interface of the subregions $i$ and $i + 1$:

$$\left( D \frac{\partial u^i}{\partial n} - u^i v^i l n \right)_{m} = \left( D \frac{\partial u^{i+1}}{\partial n} - u^{i+1} v^{i+1} l n \right)_{m}$$

Besides the above conditions, the concentration at each interface needs to be continuous, i.e.

$$u^i |_{m} = u^{i+1} |_{m}$$
After applying the boundary conditions of the problem, imposing the compatibility conditions between subdomains, rearranging all the unknowns on the left-hand side and defining a vector \( y \) on the right-hand side obtained by multiplying matrix elements by the known surface values of the concentration and its normal derivative, the following overdetermined banded linear system of equations is obtained:

\[
Ax = y
\]  

(35)

where \( x \) is the vector of unknowns \( u \) and \( q \).

We shall mention here, that for standard DRM the approximation defined in eqn (26) estimates only the values of \( u \) at the interpolation nodes, no information about the derivatives is used unless a Hermitian interpolation were implemented. Besides, the derivatives approximated by eqn (28) can differ from their exact values because the interpolation defined in eqn (26) does not use any information about the derivatives of \( u \). In the DRM-MD approach, even when non-Hermitian interpolation is used, the compatibility conditions reinforce the information on the derivatives, yielding more accurate numerical results.

The compatibility conditions are expressed numerically in the way the matrices in eqn (31) are assembled. For instance, for a domain subdivided into 4 subdomains defined by four linear continuous boundary elements (Fig. 2), the continuity of \( u \) is reflected in the assembly of the \( H \) matrix as follows

\[
\begin{align*}
H_{1}^{H} & H_{2}^{H} & 0 & 0 & 0 & 0 & 0 \\
0 & H_{2}^{H} & H_{3}^{H} & 0 & 0 & 0 & 0 \\
0 & 0 & H_{4}^{H} & H_{5}^{H} & 0 & 0 & 0 \\
0 & 0 & 0 & H_{5}^{V} & H_{6}^{V} & 0 & 0 \\
0 & 0 & 0 & 0 & H_{7}^{V} & H_{8}^{V} & 0 \\
0 & 0 & 0 & 0 & 0 & H_{9}^{V} & H_{10}^{V} \\
\end{align*}
\]  

(36)

The \( G \) matrix will be arranged in a similar fashion to comply with the antisymmetry of \( q \).
The sparsity pattern of the resulting matrix $A$ is sparse and, therefore, similar to those obtained with classical domain techniques. The more subdomains we consider, the more banded blocks are included (one block per subdomain) in the coefficient matrix. Along with this, more common interfaces are involved and in such cases the blocks overlap due to the matching conditions. Consequently the system of equations tends to be block-banded with a large number of zero submatrices. This fact leads to systems of equations with an increased sparsity pattern, as can be seen in Fig. 3.

As mentioned above, the use of the DRM-MD will increase the accuracy of the solution, compared to the single DRM, since for those internal nodes placed on
the edges of the subdomains we will be enforcing continuity and equilibrium laws. Moreover, on each subdomain the approximation will be more local, reducing the risk of propagating numerical errors.

In addition, the overall system of equations is suitable to be solved by iterative methods more efficiently due to the large number of zero entries currently present in the coefficient matrices, using similar techniques to those used in the classical domain approaches. We will further analyse this topic when we thoroughly investigate the performance of different iterative techniques together with several preconditioners.

When using continuous linear or higher order elements, the assembling procedure of the final coefficient matrix will yield to an overdetermined system of equations. In the numerical cases we will analyse, where the subdomains will all be rectangular, one degree of overdetermination is found per node shared by four subdomains.

A way to avoid this overdetermination is to make use of discontinuous boundary elements at the expense of having more equations plus more unknowns. When the complexity of the problem requires a high number of subregions, the resulting system of equations will be extremely large.

4 Approximating functions $f_j$

As pointed out above, the key feature of the DRM is the transference of domain integrals to the boundary. For this purpose it is necessary to interpolate the non-homogeneous term of the equation (forcing term) in terms of prescribed basis functions $f_j$, which are usually recommended to be RBFs.

Given a set of nodes $X = \{x_j, j = 1, 2, \ldots, N\} \in \Omega \subset \mathbb{R}^n$ a radial basis function $f_j(x) \equiv f(||x - x_j||)$ where $||x - x_j||$ is the Euclidean distance between two points or interpolation nodes. Because of RBFs’ spherical symmetry about the centres $x_j$, they are called radial. The primary advantage of RBFs is that they produce approximate functions that vary smoothly and are differentiable. Another advantage is that they involve a single independent variable regardless of the dimensionality of the problem.

The RBFs more commonly used are

$$
\begin{align*}
\phi(r) &= r^{2m-2} \log r \quad \text{(generalized thin-plate splines)}, \\
\phi(r) &= (r^2 + c^2)^{m/2} \quad \text{(generalized multiquadric)}, \\
\phi(r) &= e^{-\beta r} \quad \text{(Gaussian)}
\end{align*}
$$

where $m$ is an integer number and $r = ||x - x_j||$.

The Gaussian and the inverse multiquadric (MQ) are positive definite functions, while the thin-plate splines (TPSs) and MQ are conditionally positive definite functions of order $m$, which require the addition of a polynomial term $P_{m-1}$ of order $m - 1$ in conjunction with some homogeneous constrained conditions in order to obtain an invertible interpolation matrix.
In a typical interpolation problem we have \( N \) pairs of data points \( \{(x_j, u(x_j))\}_{j=1}^{N} \), which are assumed to be samples of some unknown function \( u \) that is to be interpolated by the function \( f \), i.e.

\[
u(x) = \sum_{j=1}^{N} \alpha_j f(\|x - x_j\|) + P_m(x) \quad x \in \mathbb{R}^2
\]  

(38)

in the sense that

\[
u(x_i) = \sum_{j=1}^{N} \alpha_j f(\|x_i - x_j\|) + P_m(x)
\]  

(39)

along with the constraints

\[
\sum_{j=1}^{N} \alpha_j P_k(x_j) = 0 \quad 1 \leq k \leq m
\]  

(40)

Here the numbers \( \alpha_j, j = 1, 2, \ldots, N \), are real coefficients and \( f \) is a radial basis function.

The matrix formulation of the above interpolation problem can be written as \( Ax = y \) with

\[
A = \begin{pmatrix} F & P_m \\ P_{m}^T & 0 \end{pmatrix}
\]  

(41)

where \( x^T(\alpha, \beta) \) and \( y^T = (u, 0) \), where \( \beta \) are the coefficients of the polynomial.

Micchelli [22] proved that for the case when the nodal points are all distinct, the matrix resulting from the above radial basis function interpolation is always nonsingular. Numerical experiments carried out by Schaback [23] proved that the condition number of matrices following on the previous interpolation, for smooth RBFs such as Gaussian or MQs, are extremely large when compared to those resulting from non-smooth RBFs such as the TPSs.

In 1982, Franke [24] published a review article assessing nearly all the interpolation methods for scattered data sets available by that time. Among the methods tested, RBFs outperformed all the other methods regarding accuracy, stability, efficiency, memory requirement and simplicity of implementation. In 1984, Stead [25] examined the accuracy of partial derivative approximations over scattered data sets, also concluding that RBFs performed more accurately compared to other considered methods. Of the RBFs tested by Franke, Hardy’s MQs [26] were positioned as the best regarding accuracy, followed by Duchon’s TPS, i.e. the generalized MQ and the TPS with \( m = 1 \) and \( m = 2 \), respectively.

Duchon [27] derived the TPS as an optimum solution to the interpolation problem in a certain Hilbert space through the construction of a reproducing kernel. Therefore, they are the natural generalization of cubic splines in \( n > 1 \) dimension. Even though the TPS have been considered as optimal interpolating multivariate functions [27] they converge linearly. The MQ functions, on the other hand, converge exponentially and always produce a minimal seminorm error [28]. Nevertheless, in
spite of this excellent performance, the MQ function contains a free parameter $c^2$ whose choice is not an easy one, and certain values can greatly affect the accuracy of the approximation.

As a result of what we discussed in this section we will present in our numerical results a comparison of the performance of the DRM-MD for both $(1 + r)$ and $(r^2 \log r + P_1)$ with $P_1 = ax + by + c$, as interpolating functions for the transfer-ence of domain integrals to the boundary.

To increase the accuracy of domain schemes which involve a domain discretization, as is the case for finite difference, finite elements, finite volumes and even the DRM-MD, a refinement of the computational mesh is required. However, in the DRM-MD accuracy can also be increased adding internal nodes to the subdomains, without the additional and non-trivial problem of further refinement of the mesh. Numerical examples comparing efficiency and accuracy of the approximated solutions for mesh refinement along with an increment of the internal DRM nodes will also be presented.

5 Iterative solution methods for sparse linear systems

5.1 Krylov subspace methods

The Krylov subspace methods are considered, currently, to be among the most important iterative techniques available for solving large linear systems. These techniques are based on projection processes, both orthogonal and oblique, onto Krylov subspaces, which are subspaces spanned by vectors of the form $p(A)v$ where $p$ is a polynomial. These techniques approximate $A^{-1}b$ by $p(A)b$, where $p$ is a good polynomial. The Krylov subspace methods are derived from, or related to, the Arnoldi orthogonalization – and this is the case for CG and GMRES – or based on Lanczos biorthogonalization.

A general projection method for solving the linear system $Ax = b$ is a method that seeks an approximate solution $x_m$ from an affine subspace $x_0 + K_m$ of dimension $m$ by imposing the condition $b - Ax_m \perp L_m$, where $L_m$ is another subspace of dimension $m$, $x_0$ being an arbitrary initial guess to the solution. A Krylov subspace method is a method for which the subspace $K_m$ is the Krylov subspace $K_m(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}$, where $r_0 = b - Ax_0$. The different versions of Krylov subspace methods arise from different choices of the subspace $L_m$ and from the ways in which the system is preconditioned.

Viewed from the angle of approximation theory, the approximations obtained from a Krylov subspace method are of the form $A^{-1}b \approx x_m = x_0 + q_{m-1}(A)r_0$, in which $q_{m-1}$ is a certain polynomial of degree $m - 1$. In the simplest case where $x_0 = 0$, then $A^{-1}b \approx q_{m-1}(A)r_0$. In other words, $A^{-1}b$ is approximated by $q_{m-1}(A)b$.

Although all techniques provide the same type of polynomial approximations, the choice of $L_m$, i.e. the constraints used to build these approximations, will have an important effect on the iterative technique. Two broad choices for $L_m$ give rise to the best-known techniques. The first is $L_m = K_m$ and the minimum residual variation $L_m = AK_m$. CG and GMRES are among the methods in this category.
The second class of methods is based on defining $L_m$ to be a Krylov subspace method associated with $A^T$, namely, $L_m = \mathcal{K}_m(A^T, r_0)$.

### 5.2 CG-type methods

CG methods can be seen as iterative solution methods to solve linear systems of equations $Ax = b$ by minimizing quadratic functionals, such as $f(x) = \frac{1}{2} x^T A x - b^T x$, if $A$ is symmetric and positive definite, or the residual functional $f(x) = (Ax - b)^T (Ax - b)$ in the general case. The minimization takes place on a sequence of subspaces $V_k$ of increasing dimension that are constructed recursively by adding a new basis vector $A^k r_0$ to those of the previous subspace $V_{k-1}$, where $V_0 = \{r_0\}$ with $r_0 = Ax_0 - b$ as the residual for the initial vector $x_0$ which can be an arbitrary approximation of $x$. Hence, $V_k$ is spanned by the vectors $r_0, A^0 r_0, \ldots, A^k r_0$.

As the Krylov vectors eventually span the whole space (or the space spanned by the eigenvectors of $A$, represented in $r_0$), the methods will give the exact solution after at most $n$ steps, where $n$ is the order of $A$. Therefore, the method can be seen as a direct solution method. However, in the presence of roundoff errors, the generated vectors will not be exactly (conjugately) orthogonal and the method may need more iterations to reach machine number precision. Even the conjugate gradient methods can generate approximations to the solution vector $x$ that are accurate after many fewer steps than $n$. The method is then used as an iterative solution method.

Conjugate gradient methods can be seen as generalized least square methods where the minimization takes place on a particular vector subspace, the Krylov subspace. Compared with other types of iterative methods, it turns out that the conjugate gradient method can converge with a faster rate, at least, when a proper preconditioning is used.

To find a minimizer of $f$, we will use an iterative method where at each stage we construct a new search direction $d^k$ (which will be conjugately orthogonal to the previous search directions). We compute the local minimizer along this search direction, i.e., given $x^k$, the approximation at stage $k$, we compute $\tau = \tau_k$ such that

$$f(x^k + \tau d^k), \quad -\infty < \tau < \infty \quad (42)$$

is minimized by $\tau_k$ and then let

$$x^{k+1} = x^k + \tau_k d^k \quad (43)$$

be the new approximation.

When the search directions are conjugately orthogonal, the residuals (or gradients) become orthogonal to the previous search directions. This property implies that the method computes the best approximation $x^{k+1} = x^k + d$ of all vectors $d$ in $V_k$.

The method whereby the search directions are computed is called the CG method. There is a somewhat simpler, but less efficient, method of computing the minimizer of $f$, called the steepest descent method. In this one the search of a local minimum takes place along the current gradient vector. On the other hand, in the CG method...
we move along a plane spanned by the gradient at the most recent point and the most recent search direction.

Because of the conjugate orthogonality property \((d_i, Ad_i) = 0, \ i \neq j\), the method was named the CG method [29]. Since it is actually the search directions that are conjugate orthogonal, a more proper name might be the conjugate direction method. However, for the inner product \((x, y) = x^T Ay\), the residuals (or gradients) indeed become conjugate orthogonal.

The CG method has the property of finite termination in the absence of roundoff errors. Nevertheless, it turns out that in practice, there is a tendency towards increasing roundoff error and loss of orthogonality. The rate of convergence depends on the distribution of eigenvalues and with a proper preconditioning of the matrix, this distribution can be such that the method converges much faster with preconditioning than without.

### 5.3 Generalized minimum residual method

The GMRES is a projection method based on taking \(K = K_m\) and \(L = AK_m\), in which \(K_m\) is the \(m\)th Krylov subspace. In the CG method, the residuals form an orthogonal basis for the space span \(\{r^0, Ar^0, A^2r^0, \ldots\}\). In GMRES, this basis is formed explicitly

\[
\begin{align*}
  w^i & = Av^i \\
  \text{do} & \quad k = 1, \ldots, i \\
  w^i & = w^i - (w^i, v^k)v^k \\
  \text{end} & \\
  v^{i+1} & = w^i/\|w^i\|
\end{align*}
\]

and all previously computed vectors in the orthogonal sequence have to be retained.

The GMRES iterates are constructed as

\[
x^i = x^0 + y_1 v^1 + \ldots + y_i v^i
\]

where the coefficients \(y_k\) have been chosen to minimize the residual norm \(\|b - Ax^i\|\).

A difficulty with the basic GMRES algorithm is that it does not provide the approximate solution \(x_m\) explicitly at each step. As a result, it is not easy to determine when to stop. One remedy is to compute the approximate solution \(x_m\) at regular intervals and check for convergence by a test on the residual. Another solution is related to the way in which the least-squares problem is solved. This approach will allow us to obtain the residual norm at every step, and a stopping criterion can be applied. If we carefully examine the GMRES algorithm, we may observe that the only possibilities of breakdown in GMRES are in the Arnoldi loop. In this situation, the algorithm stops because the next Arnoldi vector cannot be generated. However in this situation, the residual vector is zero, i.e. the algorithm will deliver the exact
solution at this step. Namely, if we have a matrix $A$ which is non-singular, then the GMRES algorithm breaks down at step $j$ if and only if the approximate solution $x_j$ is exact.

The GMRES algorithm becomes impractical when $m$ is large because of the growth of memory and computational requirements as $m$ increases. There are two remedies for this situation. One is based on restarting and the other on truncating the Arnoldi orthogonalization. The restarting GMRES consists in restarting the algorithm periodically. There are many variations to this basic scheme. One that is generally more economical in practice is based on the observation that sometimes a small $m$ is sufficient for convergence and sometimes the largest possible $m$ is necessary. A well-known difficulty with the restarted GMRES algorithm is that it can stagnate when the matrix is not positive definite. The full GMRES algorithm is guaranteed to converge in at most $n$ steps, but this would be impractical if there were many steps required for convergence. A preconditioner for the linear system can be used to reduce the number of steps, or a better preconditioner if one is already in use.

Both algorithms, GMRES and CG, are strongly related to, as well as defined by, the choice of a basis of the Krylov subspace. The GMRES algorithm uses an orthogonal basis. In the CG algorithm, the auxiliary vectors are $A$-orthogonal, i.e. conjugate.

The CG-type algorithms, i.e. algorithms defined through short-term recurrences, are more desirable than those algorithms which require storing entire sequences of vectors as in the GMRES process. On the other hand, these latter algorithms require less memory and operations per step.

### 5.3.1 Convergence of the CG and GMRES algorithms and eigenvalue distribution

The CG and GMRES algorithms, each generate the optimal approximate solution from a Krylov subspace, where optimal means to have an error with minimal $A$-norm in the case of CG or having a residual with minimal 2-norm in the case of GMRES. To describe the behaviour of the algorithms it is appropriate to derive a sharp upper bound on the reduction in the $A$-norm of the error for CG or in the 2-norm of the residual for GMRES, i.e. an upper bound that is independent of the initial vector but that is actually attained for certain initial vectors.

CG-type methods consist mainly of matrix vector multiplication and vector operations. If the matrix is normal, then the norm of the residual can be estimated by the error in a best polynomial approximation problem. The rate of convergence depends on the distribution of eigenvalues and, to some extent, also on the initial residual. If the matrix is not normal, as it is the case in DRM-MD, the estimate also involves the condition number of the matrix. In general, we are unable to give a sufficiently accurate estimate, but we can derive various upper bounds.

For Hermitian problems, the $A$-norm of the error in the CG algorithm is minimized over the space $e_0 + \text{span}[ Ae_0, A^2e_0, \ldots, A^ke_0]$. The CG error vector at step $k$ can be written as a function of $e_0$ and $k$th-degree polynomials with value 1 at the origin.
In other words, the error $e_k$ in the CG approximation satisfies
\[ \|e_k\|_A = \min_{p_k} \|p_k(A)e_0\|_A \]
where the minimum is taken over all polynomials $p_k$ of degree $k$ or less with $p_k(0) = 1$.

A sharp upper bound is derived involving all of the eigenvalues of $A$. This sharp upper bound can be written in the form
\[ \|e_k\|_A / \|e_0\|_A \leq \min_{p_k} \max_{i=1,...,n} |p_k(\lambda_i)| \quad \text{for CG} \]

The problem of describing the convergence of this algorithm, therefore, reduces to one in approximation theory – how well can we approximate zero on the set of eigenvalues of $A$ using a $k$th-degree polynomial with value 1 at the origin. This minimax polynomial can be calculated if the eigenvalues of $A$ are known; more importantly, this sharp upper bound provides an intuition as to what constitutes good and bad eigenvalue distributions. Eigenvalues tightly clustered around a single point (away from the origin) are good. Widely spread eigenvalues, especially if they lie on both sides of the origin, are bad, because a low-degree polynomial with value 1 at the origin cannot be small at a large number of such points.

When we have only limited information about the eigenvalues of $A$, it is useful to have error bounds that involve only a few properties of the eigenvalues. For example, in the CG algorithm for Hermitian positive definite problems, knowing only the largest and smallest eigenvalues of $A$, we can obtain an error bound by considering the minimax polynomial on the interval from $\lambda_{\min}$ to $\lambda_{\max}$.

Knowing only the largest and smallest eigenvalues of a Hermitian positive definite matrix $A$, the best possible bound is
\[ \frac{\|e_k\|_A}{\|e_0\|_A} \leq 2 \left( \frac{\sqrt{k-1}}{\sqrt{k+1}} \right)^{-1} \leq 2 \left( \frac{\sqrt{k-1}}{\sqrt{k+1}} \right) \]
where $k = \lambda_{\max}/\lambda_{\min}$ is the ratio of the largest to smallest eigenvalue of $A$.

If additional information is available about the interior eigenvalues of $A$, we can often improve on the estimate while maintaining a simpler expression than the sharp bound (45). For example, when $A$ has one eigenvalue much larger than the others, say, $\lambda_1 \leq \cdots \leq \lambda_{n-1} \ll \lambda_n$, that is, $\lambda_n/\lambda_{n-1} \gg 1$, it follows that
\[ \frac{\|e_k\|_A}{\|e_0\|_A} \leq 2 \left( \frac{\sqrt{k_{n-1}-1}}{\sqrt{k_{n-1}+1}} \right)^{k-1} , \quad k_{n-1} = \frac{\lambda_{n-1}}{\lambda_n} \]

Similarly if the matrix $A$ has just a few large outlying eigenvalues, i.e. $\lambda_1 \leq \cdots \leq \lambda_{n-l} \ll \lambda_{n-l+1} \leq \cdots \leq \lambda_n$ ($\lambda_{n-l+1}/\lambda_{n-l} \gg 1$)
\[ \frac{\|e_k\|_A}{\|e_0\|_A} \leq 2 \left( \frac{\sqrt{k_{n-l}-1}}{\sqrt{k_{n-l}+1}} \right)^{k-l} , \quad k_{n-l} = \frac{\lambda_{n-l}}{\lambda_n} \]
The rate of convergence can be measured in various norms. When a relative measure is used – say, the ratio of the norm of the current error and the initial error – the condition number of the matrix can frequently be used to give a sufficiently accurate estimate. More generally, if the eigenvalues are contained in an ellipse not covering the origin and with sufficiently small eccentricity, then similar estimates can be derived.

However, when the eigenvalues are distributed non-uniformly in this interval or ellipse, the condition number alone gives too rough an overestimate of the necessary number of iterations. Therefore, it is possible to find various improved estimates of the number of iterations required to get a sufficiently small relative residual, based on some additional assumptions made of the eigenvalue distribution. In particular, that could be the case of isolated eigenvalues at one or both ends of the spectrum.

Regarding GMRES, for \( x_k \) being the approximate solution from the \( k \)th step of the algorithm, and the \( k \)th step residual \( r_k = b - Ax_k \). Then, the approximation obtained is \( x_k = x_0 + q_k(A)r_0 \) where \( q_k \) is a polynomial of degree \( k - 1 \) such that

\[
\|r_k\|_2 = \|(I - Aq_k(A))r_0\|_2 = \min_{q \in \mathbb{P}_{k-1}} \|(I - Aq(A))r_0\|_2
\] (49)

Unfortunately, it is not possible to get a simple result for this expression unless the matrix \( A \) is normal.

If we assume that \( A \) is a diagonalizable matrix \( A = X \Lambda X^{-1} \), where \( \Lambda = \text{diag} \{\lambda_1, \lambda_2, \ldots, \lambda_n\} \) is the diagonal matrix of eigenvalues, and we define

\[
e^{(k)} = \min_{p \in \mathbb{P}_k, p(0)=1} \max_{i=1,\ldots,n} |p(\lambda_i)|
\]

Then, again, the rate of convergence of the GMRES algorithm can be measured through the residual norm achieved by the \( k \)th step of the algorithm. Satisfying the inequality \( \|r_k\|_2 \leq \kappa_2(X)e^{(k)}\|r_0\|_2 \), where \( \kappa_2(X) = \|X\|_2\|X^{-1}\|_2 \), \( \kappa_2 \) is the condition number of the matrix of eigenvectors.

In this case, as in the Hermitian case, the problem of describing the convergence of GMRES reduces to a problem in approximation theory – how well we can approximate zero on the set of complex eigenvalues using a \( k \)th-degree polynomial with value 1 at the origin. We do not have simple estimates based on the ratio of largest to smallest eigenvalues, but we can still apply some notions about good and bad eigenvalue distributions in the complex plane. Eigenvalues tightly clustered about a single point, away from the origin are good, since, for instance, the polynomial \( (1 - z/c)^k \) is small at all points close to \( c \) in the complex plane. Eigenvalues all around the origin are bad because, by the maximum principle, it is impossible to have a polynomial that is 1 at the origin and small in absolute value at many points distributed all around the origin. In this case, it is \( A \)’s eigenvalue distribution that essentially determines the behaviour of GMRES.

From what we have said we find a close relationship between condition numbers, distribution of eigenvalues and rate of convergence of the CG-like and GMRES algorithms. For this reason we will present in the numerical analysis a report of the eigenvalue (singular values when \( A \) is overdetermined) distribution of the different
A matrices arising from different discretizations of the domain after the treatment of the studied problem with the DRM-MD. In addition we will describe the performance of CG-like and GMRES methods with and without preconditioners.

5.3.2 Methods related to the normal equation
As we stated in Section 3.2, the system of equations arising from the DRM-MD approach is overdetermined. There are a number of techniques for converting an overdetermined and non-symmetric linear system of equations into a closed and symmetric one. One such technique solves the equivalent linear system $A^T Ax = A^T b$, called the normal equations. Often, this approach is avoided in practice because the coefficient matrix $A^T A$ is much worse conditioned than $A$. However, the normal equations approach may be adequate in some situations. There are certain applications in which it is preferred to the usual Krylov subspace techniques.

A general consensus is that solving the normal equations can be an inefficient approach in the case when $A$ is poorly conditioned. Indeed, the 2-norm condition number of $A^T A$ is given by $\text{Cond}_2(A^T A) = \|A^T A\|_2 \| (A^T A)^{-1} \|_2$.

$$\|A^T A\|_2 = \sigma_{\text{max}}(A)$$ where $\sigma_{\text{max}}(A)$ is the largest singular value of $A$, which, incidentally, is also equal to the 2-norm of $A$. Thus, using a similar argument for the inverse $(A^T A)^{-1}$ yields $\text{Cond}_2(A^T A) = \|A\|_2^2 \|A^{-1}\|_2^2 = \text{Cond}_2^2(A)$.

The 2-norm condition number for $A^T A$ is exactly the square of the condition number of $A$, which could cause difficulties. For example, if originally $\text{Cond}_2(A) = 10^8$, then an iterative method may be able to perform reasonably well. However, a condition number of $10^{16}$ can be much more difficult to handle by a standard iterative method. That is because any progress made in one step of the iterative procedure may be annihilated by the noise due to numerical errors. On the other hand, if the original matrix has a good 2-norm condition number, then the normal equation approach should not cause any serious difficulties.

The normal equations approach might be better, because of its robust quality (symmetric and positive definite matrices) which outweighs the additional cost due to the slowness of the method in the generic case. For ill-conditioned problems, these methods will simply fail to converge, unless a good preconditioner is available.

In the numerical results we will present singular-eigenvalue analysis to determine whether it is suitable to use the normal equation as input data for the iterative techniques, as well as those iterative algorithms where the Krylov subspace is associated with $A^T$.

5.4 LSQR, algorithm for sparse least-squares problems
The LSQR is an iterative method for computing a solution $x$ to either unsymmetric equations $Ax = b$ or linear least squares $\min \|Ax - b\|_2$. Developed by Paige and Saunders in 1982 [8], it is analytically equivalent to the standard method of conjugate gradients, but with more favourable numerical properties.

CG-like methods are characterized by their need for only a few vectors of working storage and by their theoretical convergence within at most $n$ iterations (if exact
arithmetic could be performed). And they are, in general, more useful when \( A \) is well-conditioned and has many nearly equal singular values.

The LSQR is based on the bidiagonalization procedure of Golub and Kahan [30]. It generates a sequence of approximations \( \{x_k\} \) such that the residual norm \( \|r_k\|_2 \) decreases monotonically, where \( r_k = b - Ax_k \). Analytically, the sequence \( \{x_k\} \) is identical to the sequence generated by the standard CG algorithm.

The algorithm is derived by applying the Lanczos process to a particular symmetric system. Given a symmetric matrix \( B \) and a starting vector \( b \), the Lanczos process is a method for generating a sequence of vectors \( \{v_i\} \) and scalars \( \{\alpha_i\}, \{\beta_i\} \) such that \( B \) is reduced to tridiagonal form. After applying \( k \) steps of the Lanczos process we reach

\[
BV_k = V_k T_k + \beta_{k+1} v_{k+1} e_k^T
\]

where \( T_k \equiv \text{tridiag}(\beta_i, \alpha_i, \beta_{i+1}) \) and \( V_k \equiv [v_1, v_2, \ldots, v_k] \). In the event we wish to solve a symmetric system \( Bx = b \), multiplying (50) by an arbitrary \( k \)-vector \( y_k \) gives

\[
BV_k y_k = V_k T_k y_k + \beta_{k+1} v_{k+1} y_k, \quad \text{where } \eta_k \text{ is the last element of } y_k.
\]

If \( y_k \) and \( x_k \) are defined by the equations

\[
T_k y_k = \beta_1 e_1 \quad x_k = V_k y_k
\]

then we shall have \( B x_k = b + \eta_k \beta_{k+1} v_{k+1} \). Hence, \( x_k \) may be taken as the exact solution to a perturbed system and will solve the original system whenever \( \eta_k \beta_{k+1} \) is negligibly small.

### 5.4.1 The least-squares system

When the Lanczos process is applied to the symmetric system that satisfies the least-squares problem, it results in two forms of a bidiagonalization procedures, depending on the starting vector.

- **Bidiag 1**: starting vector \( b \); reduction to lower bidiagonal form \( \rightarrow A^T U_{k+1} = V_{k} B_k^T + \alpha_{k+1} v_{k+1} e_k^T + 1 \)
- **Bidiag 2**: starting vector \( A^T b \); reduction to upper bidiagonal form \( \rightarrow A^T P_k = V_{k} R_k^T + \theta_{k+1} v_{k+1} e_k^T + 1 \)

The principal connection between the two bidiagonalization procedures is that the matrices \( V_k \) are the same for each, and that the identity

\[
B_k^T B_k = R_k^T R_k
\]

holds. This follows from the fact that \( v_1 \) is the same in both cases, and \( V_k \) is the result of applying the Lanczos process 50 with \( B = A^T A \). The rather surprising conclusion is that \( R_k \) must be identical to the matrix that would be obtained from the conventional QR factorization of \( B_k \). Thus

\[
Q_k B_k = R_k 0
\]
The quantities generated from \( A \) and \( b \) by Bidiag 1 is used to solve the least-squares problem, \( \min \| b - Ax \| \). Computationally, it is advantageous to solve this problem using the standard QR factorization of \( B_k \), that is, the same factorization 51 that links the two bidiagonalizations. LSQR needs to save only the most recent iterates.

The stopping criteria is set in terms of three dimensionless quantities, which the user is required to specify. These rules are based on allowable perturbations in the data, and in an attempt to regularize ill-conditioned systems. Stability is more apparent for LSQR than for the standard method of conjugate gradients.

After having performed various numerical comparisons, the authors of LSQR recommend the application of the symmetric CG to the normal equations \( A^T Ax = A^T b \) only if it would produce a satisfactory estimate of \( x \) in very few iterations.

For ill-conditioned matrices, LSQR should be more reliable than the least-squares adaption of symmetric CG, at the expense of more storage and work per iteration.

### 5.4.2 Preconditioned iterations

Although iterative methods are well founded theoretically, they are all likely to suffer from slow convergence for problems which arise from typical applications. Preconditioning is a key factor for the success of the Krylov subspace methods in these applications.

A weakness of iterative solvers, compared to direct solvers, is the lack of robustness they may have. Both the efficiency and robustness of iterative techniques can be improved by using **preconditioning**. Preconditioning is a means of transforming the original linear system into one which has the same solution, but which is likely to be easier to solve with an iterative solver. In general, the reliability of the iterative techniques, when dealing with various applications, depends much more on the quality of the preconditioner than on the particular Krylov subspace used.

Finding a good preconditioner to solve a given sparse linear system is rarely based on theoretical results, and some methods work surprisingly well, often despite expectations. A preconditioner can be defined as any subsidiary approximate solver which is combined with an outer iteration technique, typically one of the Krylov subspace iterations. We have to note that there are virtually no limits to available options for obtaining good preconditioners. For example, preconditioners can be derived from the knowledge of the original physical problems from which the linear system arises. However, a common feature of the preconditioners is that they are built from the original coefficient matrix.

In general, we can say that a preconditioner is any form of implicit or explicit modification of an original linear system which makes it easier to solve by a given iterative method. For example, scaling all rows of a linear system to make the diagonal elements equal to one is an explicit form of preconditioning. The resulting system can be solved by a Krylov subspace method and may require fewer steps to converge than with the original system (although this is not guaranteed).

Then a preconditioner \( M \) is a matrix which approximates the system matrix \( A \) in some sense. One of the simplest ways of defining a preconditioner is to assume that \( M \) is a diagonal matrix with \( A \)'s main diagonal as entries. This preconditioner is
called *Jacobi* preconditioner. From a practical point of view, the only requirement for $M$ is that it is inexpensive to solve linear systems $Mx = b$. This is because the preconditioned algorithms will all require a linear system solution with the matrix $M$ at each step. Then, the following preconditioned system could be solved: $M^{-1}Ax = M^{-1}b$ or $AM^{-1}u = b$, $x = M^{-1}u$ (left or right preconditioning, respectively). In theory, any general splitting in which $M$ is nonsingular can be used.

In the case of preconditioned CG, left and right preconditioning lead to two systems of equations which are no longer symmetric in general.

When $M$ is available in the form of an incomplete factorization, i.e. when $M = LL^T$, then a simple way to preserve symmetry is to split the preconditioner between left and right, i.e. to solve $L^{-1}AL^{-T}u = L^{-1}b$, $x = L^{-T}u$, which involves a symmetric positive definite matrix.

However, it is not necessary to split the preconditioner in this manner in order to preserve symmetry. We can observe that $M^{-1}A$ is self-adjoint for the $M$-inner product, $(x, y)_M = (Ax, y)$, since $(M^{-1}Ax, y)_M = (Ax, y) = (x, M(M^{-1}Ay))_M$. Therefore, an alternative is to replace the usual Euclidean inner product in the CG algorithm by the $M$-inner product.

It is interesting to observe that $M^{-1}A$ is also self-adjoint with respect to the $A$-inner product. Indeed, $(M^{-1}Ax, y)_A = (AM^{-1}Ax, y) = (x, AM^{-1}Ay) = (x, M^{-1}Ay)_A$ and a similar algorithm can be written for this dot product.

When $M$ is a Cholesky product $M = LL^T$, two options are available, namely, the split preconditioning option, which is the one we present in the numerical results; or the algorithm using the $M$-inner product (left preconditioned system).

If we consider a right preconditioned system, the matrix $AM^{-1}$ is not Hermitian with either the standard inner product or the $M$-inner product. However, it is Hermitian with respect to the $M^{-1}$-inner product. If we introduce this $M^{-1}$-inner product in the CG algorithm we get the right-preconditioned CG.

The implication is that the left-preconditioned CG algorithm with the $M$-inner product is mathematically equivalent to the right-preconditioned CG algorithm with the $M^{-1}$-inner product.

In the case of preconditioned GMRES the same three options for applying the preconditioning operation as for the CG (namely, left, split and right preconditioning) are available. However, there will be one fundamental difference – the right preconditioning versions will give rise to what is called a flexible variant, i.e. a variant in which the preconditioner can change at each step.

In many cases, $M$ is the result of a factorization of the form $M = LU$. Then there is the option of using GMRES on the split-preconditioned system $L^{-1}AU^{-1}u = L^{-1}b$, $x = U^{-1}u$. In this situation the residual norm available is that of $L^{-1}(b - Ax_m)$.

The question arising on the differences between the right, left and split preconditioning is the fact that different versions of the residuals are available in each case. And this may affect the stopping criterion and may cause the algorithm to stop either prematurely or with delay.

When comparing the left, right and split preconditioning options, a first observation to make is that the spectra of the three associated operators $M^{-1}A$, $AM^{-1}$...
and \( L^{-1}AU^{-1} \) are identical. Therefore, in principle one should expect convergence to be similar, although eigenvalues do not always govern convergence.

In most practical situations, the comparison between the difference in the convergence behaviour of right and left preconditioning is not significant. The only exception is when \( M \) is ill-conditioned which could lead to substantial differences if we consider that in the right preconditioning case the residual norm is \( \| b - Ax_m \|_2 \), and the preconditioned residual norm in the left preconditioning case is \( \| M^{-1}(b - Ax_m) \|_2 \). It is for this reason that we will be solving the system with right preconditioning in the numerical results. In Table 1 we present a comparison of the condition numbers of the normal equation \( A^T A \), and the condition number \( (cn) \) of the preconditioner matrix ILU of the numerical examples presented in Section 6, using different subregions and interpolation functions. For some of the numerical examples considered, as can be observed, the \( cn \)s of the normal equation are not significantly large, but when the \( cn \)s of \( M \) are too large it is not possible to solve the preconditioned systems of equations.

When \( M \) is available in the form of an incomplete factorization of the original matrix \( A \), of the form \( A = LU - R \) where \( L \) and \( U \) have the same non-zero structure as the lower and upper parts of \( A \), respectively, and \( R \) is the residual or error of the factorization. This incomplete factorization is rather easy and inexpensive to compute. On the other hand, it often leads to a crude approximation which may result in the Krylov subspace accelerator requiring many iterations to converge. To remedy this, several alternative incomplete factorizations have been developed by allowing more fill-in in \( L \) and \( U \). In general, the more accurate ILU factorizations require fewer iterations to converge, but the preprocessing cost to compute the factors is higher.

### Table 1: Condition numbers of ILU preconditioners and their applicability.

<table>
<thead>
<tr>
<th>Mesh ( A(m, n) ) – RBF</th>
<th>Condition number ( A^T A )</th>
<th>Condition number ( M )</th>
<th>A/NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mesh I ((400, 343)) ((1 + r))</td>
<td>3.2921E+05</td>
<td>4.1928E+07</td>
<td>A</td>
</tr>
<tr>
<td>Mesh I ((400, 343)) ((r^2 \log r + p))</td>
<td>6.9687E+05</td>
<td>5.6078E+08</td>
<td>A</td>
</tr>
<tr>
<td>Mesh II ((472, 415)) ((1 + r))</td>
<td>7.2188E+05</td>
<td>1.6735E+13</td>
<td>A</td>
</tr>
<tr>
<td>Mesh II ((472, 415)) ((r^2 \log r + p))</td>
<td>1.6265E+06</td>
<td>6.4587E+19</td>
<td>NA</td>
</tr>
<tr>
<td>Mesh III ((512, 454)) ((1 + r))</td>
<td>8.3502E+05</td>
<td>2.2552E+18</td>
<td>A</td>
</tr>
<tr>
<td>Mesh III ((512, 454)) ((r^2 \log r + p))</td>
<td>2.2670E+06</td>
<td>4.0696E+20</td>
<td>NA</td>
</tr>
<tr>
<td>Mesh IV ((720, 663)) ((1 + r))</td>
<td>7.0963E+05</td>
<td>5.0890E+19</td>
<td>NA</td>
</tr>
<tr>
<td>Mesh IV ((720, 663)) ((r^2 \log r + p))</td>
<td>4.5261E+06</td>
<td>1.3739E+20</td>
<td>NA</td>
</tr>
<tr>
<td>Mesh V ((1000, 853)) ((1 + r))</td>
<td>1.4099E+06</td>
<td>4.1119E+28</td>
<td>NA</td>
</tr>
<tr>
<td>Mesh V ((1000, 853)) ((r^2 \log r + p))</td>
<td>2.4351E+06</td>
<td>6.3813E+21</td>
<td>NA</td>
</tr>
<tr>
<td>Mesh VI ((1000, 853)) ((1 + r))</td>
<td>4.4126E+05</td>
<td>1.1470E+09</td>
<td>A</td>
</tr>
<tr>
<td>Mesh VI ((1000, 853)) ((r^2 \log r + p))</td>
<td>1.0155E+06</td>
<td>9.7157E+10</td>
<td>A</td>
</tr>
</tbody>
</table>

A, applicable; NA, not applicable.
When the preconditioning matrix is of the form \( M = LU \) where \( L \) and \( U \) have the same pattern as the \( L \)-part and the \( U \)-part of \( A \), respectively, the question is whether or not it is possible to find \( L \) and \( U \) that yield an error that is smaller in some sense than the one we get from the same method without any preconditioning. We can, for example, try to find such an incomplete factorization in which the residual matrix \( A - LU \) has zero elements in locations where \( A \) has non-zero entries. This turns out to be possible in general and yields the ILU(0) factorization. Generally, a pattern for \( L \) and \( U \) can be specified and \( L \) and \( U \) may be sought so that they satisfy certain conditions. This leads to the general class of incomplete factorization techniques.

5.4.3 Level of fill and ILU threshold strategies

A general Incomplete LU factorization can be derived by performing Gaussian elimination and dropping some elements in predetermined non-diagonal positions. We can consider any lower triangular matrix \( L \) which has the same structure as the lower part of \( A \), and any matrix \( U \) which has the same structure as that of the upper part of \( A \). If the product \( LU \) were performed, the resulting matrix would not have the same pattern of \( A \), due to the extra diagonals in the product. It is impossible, in general, to match \( A \) with this product for any \( L \) and \( U \). The entries in these extra diagonals are called fill-in elements.

In domain-dominant problems, as are the problems to be solved by the DRM-MD approach, the order of the system matrix is very large (being dependant on the number of subdomains and the number of DRM nodes in every subdomain). Still, the matrix has a sparse structure, meaning that most entries \( a_{ij} \) are zero. However, during the factorization of \( A \), entries that are zero can be replaced by non-zero at later stages (fill-in elements). If these fill-in elements are ignored, then it is possible to find \( L \) and \( U \) so that their product is equal to \( A \) in the other diagonals. This defines the ILU(0) factorization, in general terms: any pair of matrices \( L \) (unit lower triangular) and \( U \) (upper triangular) so that the elements of \( A - LU \) are zero in the locations of NZ(\( A \)). The accuracy of the ILU(0) incomplete factorization may be insufficient to yield an adequate rate of convergence. More accurate Incomplete LU factorizations are often more efficient as well as more reliable. These more accurate factorizations will differ from ILU(0) by allowing some fill-in.

Incomplete factorizations which rely on the levels of fill are blind to numerical values because elements that are dropped depend only on the structure of \( A \). A generic ILU algorithm with threshold (ILUT) can be derived by including a set of rules for dropping small elements [31]. Applying a dropping rule to an element may only mean replacing the element by zero if it satisfies a set of criteria. The dropping rule for ILU(0) is to drop elements that are in positions not belonging to the original structure of the matrix.

In the factorization ILUT(\( p, \tau \)), the following procedure is accomplished: an element \( w_k \) is dropped (i.e. replaced by zero) if it is less than the relative tolerance \( \tau_i \) obtained by multiplying \( \tau \) by the original norm of the \( i \)th row (e.g. the 2-norm). Then, a dropping rule of a different type is applied. First, drop again any element in the row with a magnitude that is below the relative tolerance \( \tau_i \). Then, keep only
the $p$ largest elements in the $L$ part of the row and the $p$ largest elements in the $U$ part of the row in addition to the diagonal element, which is always kept.

The goal of the second dropping step is to control the number of elements per row. Here, $p$ can be viewed as a parameter that helps control memory usage, while $\tau$ helps to reduce computational cost. There are several possible variations on the implementation of the second dropping step. For example we can keep a number of elements equal to $nu(i) + p$ in the upper part and $nl(i) + p$ in the lower part of the row, where $nl(i)$ and $nu(i)$ are the number of non-zero elements in the $L$ part and the $U$ part of the $i$th row of $A$, respectively. This variant is adopted in the ILUT code used for the numerical examples.

The changes in the sparsity pattern from a matrix $A(400, 343)$ arising from DRM-MD, towards the preconditioned ILUT matrix used as input for the GMRES algorithm are shown in Fig. 4.

![Figure 4: (a) Matrix $A(m, n)$; (b) matrix $A^T A(n, n)$; (c) preconditioning matrix $M = ILUT(n, n)$; (d) left preconditioned matrix $[M^{-1}(A^T A)](n, n)$.](image-url)
5.4.4 Preconditioned LSQR

The number of iterations required for the LSQR algorithm to reach a certain accuracy depends strongly on the scaling of the problem. Poor scaling of the rows or columns of \( A \) would lead to inefficient convergence. For example if a row of \( A \) is very small or large compared to the other rows of \( A \), the corresponding row of \( A \) and \( b \) should be scaled up or down. Unless better information is known, the non-zero columns of \( A \) should be scaled so that they all have the same Euclidean norm (e.g. 1.0).

Another way to reduce the number of iterations is to solve a related system \( Mx = b \) efficiently, where \( M \) approximates \( A \) in some helpful way (e.g. \( M - A \) has low rank or its elements are small relative to those of \( A \)). LSQR may converge more rapidly on the system \( AM^{-1}u = b \), after which \( x \) can be recovered by solving \( x = M^{-1}u \).

In the numerical analysis we will report the behaviour of LSQR with Jacobian preconditioner and with column scaling.

6 Numerical analysis

To test the performance of the different iterative solvers as well as to observe the behaviour of the numerical solution when the two different interpolation functions are used, we will consider a one-dimensional convection–diffusion problem with variable velocity. This equation has been used before as a test example of different implementations in the DRM literature (see [20, 32–34]).

In the problem under consideration the convective velocity is assumed to be a linear function of the longitudinal direction and in the governing equation the constant decay parameter is related to the gradient of the velocity in the following way:

\[
D \frac{d^2 u}{dx^2} - (A + kx) \frac{du}{dx} - ku = 0
\]  

(52)

Dirichlet boundary conditions at the inlet and outlet of the domain are imposed, i.e. \( u(0) = U_0 \) and \( u(L) = U_1 \). In the particular case when \( D = 1, L = 1 \) and \( A = (\ln(U_1/U_0) - k/2) \), a simple analytical solution can be obtained:

\[
u = U_0 \exp \left\{ k \frac{x^2}{2} + (\ln \frac{U_1}{U_0} - k \frac{1}{2}) x \right\}
\]

(53)

This simplified case is the one that has been used before as a benchmark test in the DRM literature. In our numerical analysis we will deal with this one-dimensional problem as a two-dimensional one in a rectangular domain with dimension \( 1 \times 0.2 \), where \( 0 \leq x \leq 1 \) and \(-0.1 \leq y \leq 0.1 \), the value considered for the decay parameter is \( k = 40 \).
The boundary conditions of our numerical problem are

\[
\begin{align*}
    u(0, y) &= U_0 = 300; \\
    u(1, y) &= U_1 = 10 \\
    \frac{\partial u}{\partial n} |_{y=-0.1} &= 0; \\
    \frac{\partial u}{\partial n} |_{y=0.1} &= 0
\end{align*}
\]

In Fig. 5 we show the analytical solution for the concentration for the studied problem with the above boundary conditions. In the analysis of the results we will present the profiles of the concentration in semi-logarithmic scale to be able to appreciate in more detail the differences between the analytical and the approximated solutions.

For the numerical comparison of the results, accuracy and efficiency are of primary interest. Accuracy is related to several concepts, consistency will be tested through the results for six different subdivisions of the domain (approximation power) and stability will be considered as we report the relative errors for the different meshes and different iterative methods. Regarding efficiency we will study the performance of the iterative methods, along with several chosen preconditioners to accelerate their rate of convergence, in terms of the CPU time and number of iterations.

To show the influence of the distribution of the internal nodes in the DRM-MD we present results for four meshes with the same grid but varying the distribution of internal nodes in each subdomain. Besides, we also study two further meshes with a bigger number of subdomains, one with a uniform distribution of subdomains and the other with non-uniform ones.

Mesh I (Fig. 6a): with 1 central internal node per subdomain, resulting in an overall mesh of 80 cells and 185 nodes.

![Figure 5: Analytical solution for the concentration.](image-url)
Mesh II (Fig. 6b): 1, 2 or 4 internal nodes per subdomain distributed in a way that more internal nodes are used in those subregions where the bigger errors were obtained with the previous mesh. Resulting in an overall mesh of 80 cells and 257 nodes.

Mesh III (Fig. 7a): 1, 3 or 5 internal nodes per subdomain. Here every subdomain has the same distribution of internal nodes as mesh II, plus its central point as an internal node, resulting in an overall mesh with 80 cells and 297 nodes.

Mesh IV (Fig. 7b): 5 internal nodes per subdomain. An overall mesh with 80 cells and 505 nodes.
Mesh V (Fig. 8a): a mesh with 200 cells, increasing the density of the grid in the centre of the domain (opposite to the previous meshes, where the grid was more dense towards the ends of the domain), and with 1 central internal node per subdomain, resulting in an overall mesh of 200 cells and 455 nodes.

Mesh VI (Fig. 8b): a mesh with 200 cells, uniformly distributed in a regular grid; with 1 central internal node per subdomain. Resulting in an overall mesh of 200 cells and 455 nodes.

As we have stated in Section 4, the six mentioned meshes are tested for \((1 + r)\) and \((r^2 \log r + P_1)\) as interpolating radial functions.
6.1 Computer implementation

If two iterative methods are both capable of generating a sufficiently accurate approximation to a system of linear equations, then we usually compare the two methods by counting operations (how many additions, subtractions, multiplications and divisions each method requires). If the number of operations per iteration is about the same for the two methods, then we might just compare the number of iterations. We should be very careful about iteration count comparisons, to be sure that the algorithms being compared really do require the same amount of work per
Domain Decomposition Techniques for Boundary Elements

iteration. The bulk of the time for the iterative solver lies in the matrix–vector multiplication and in the preconditioning step. All the algorithms that we implemented perform matrix–vector multiplications, as well as vector inner products during the course of the iteration. This is why all the matrices were stored using the same storage scheme, which allows the operations to be comparable. In the tables we present regarding efficiency we report CPU time, in addition to the number of iterations.

All the algorithms were implemented in Fortran using double precision arithmetic. Test runs were performed on a Pentium IV with one 2.53 GHz processor.

6.2 Performance of Krylov iterative solvers

6.2.1 CG scheme

A CG algorithm for non-symmetric systems (CGN) was tested for the over-determined matrix obtained from the DRM-MD approach. As the original residual \( r_i = b - Ax \) must be available at every step, we compute the residual in two parts: \( r_{i+1} = r_i - \alpha A p_i \) and then \( A^T r_{i+1} \) which is the residual for the normal equation \( A^T A x = A^T b \). In this case, \( x_m \) minimizes the function \( f(x) = (A^T A(x_m - x), (x_m, x)) \) (the difference with the GMRES algorithm is the subspace in which the residual norm is minimized), while \( A^T A \) is never formed in practice.

The convergence behaviour of CGN was often quite irregular, leading to a loss of accuracy. The performance improved when a Jacobi preconditioner was applied, after a partial pivoting being implemented to get rid of the zero entries in the main diagonal. As for efficiency, the values of CPU time were one (and sometimes even two) orders of magnitude higher than the values obtained for the CG algorithm used with the explicit normal system of equations. As the results do not compare well against the analytical solution we only present those obtained with the symmetric CG method.

To determine whether the use of the explicit normal equation on Krylov solvers is suitable, rather than the direct equation, we perform a singular/eigenvalue analysis. From the comparison of the eigenvalue distribution of \( A^T A \) against the distribution of singular values of \( A \) for the different meshes, we observe that the eigenvalues follow the singular values quite closely. This tendency is increased when the mesh has more internal nodes per subdomain, as we can see in Fig. 9 (mesh I with one internal node per subdomain, mesh IV with five internal nodes per subdomain). In both Fig. 9a and b we can see some scattered eigenvalues in the upper end of the spectrum, but still most of the eigenvalues have a good matching with the singular values. The spectra for meshes II and III are in between those shown in Fig. 9, while the distributions for meshes V and VI follow the same pattern as mesh I (due to also having one internal node per subdomain).

In Fig. 9 we show results for the TPS interpolation function, similar behaviour is found for the interpolation function \((1 + r)\).

Thus, when we apply the CG algorithm to the normal equation (the algebraic system is symmetric and positive definite), it performs well when we compare the results with the analytical solution of the studied problem. However, when we tested preconditioned versions of CG, the algorithm fails to converge in less than
the expected steps for the applied preconditioners. Failures of this type, characterized by very slow convergence, are rather common for CG. One reason of the poor performance of preconditioned CG is that even if $M$ and $A$ are symmetric and positive definite, generally $M^{-1}A$ is not. Another drawback with CG is the effect of finite precision. In fact, the basis vectors generated by this algorithm in finite precision arithmetic often loses orthogonality completely and may even become linearly dependent, especially when the system is ill-conditioned as it usually happens with normal equations. This situation frequently takes place with the CG algorithm, which uses a short recurrence to generate orthogonal basis vectors for the Krylov subspace [35, 36].

In Fig. 10 we show the concentration profile obtained for the set of six meshes already described; Fig. 10a corresponds to the interpolation function $(1 + r)$ and Fig. 10b to $(r^2 \log r + P_1)$. The corresponding relative errors are presented in Fig. 11. In Table 2 we report the maximum relative errors (%) for the concentration for the two interpolation functions. As can be observed, in this case the best solution is obtained with the mesh IV using the ATPS function and without preconditioner. Notice that the maximum relative error of 11% is found in the region where the analytical solution of the problem is almost zero. The accuracy reached by meshes V and VI (200 subdomains) deteriorate with respect to mesh IV (80 subdomains).
To analyse the efficiency of the CG algorithm we will compare the number of iterations and the cpu time required to reach the approximate solution for the six different meshes. In terms of storage requirements, the model size grows with the number of total nodes of the mesh, and consequently the number of iterations and cpu time does (see Table 3). Comparing running time for the same mesh and different interpolation functions, we observe that the cpu time and number
Figure 11: Relative error for the concentration for the interpolation functions (a) \((1 + r)\) and (b) \((r^2 \log r + P_1)\).

of iterations are higher for the function \((r^2 \log r + P_1)\) (Table 3). As we stated in Section 5.3.1, the convergence of CG is related to the condition number and the distribution of the eigenvalues of \(A^T A\). Analyzing then the eigenvalue distribution for the six meshes and for the two interpolation functions, we find that for the function \((1 + r)\) the eigenvalues are evenly distributed between \(\lambda_{\text{min}}\) and \(\lambda_{\text{max}}\), whereas for \((r^2 \log r + P_1)\) the interval spanned by \(\lambda_{\text{min}}\) and \(\lambda_{\text{max}}\) is wider (meaning a higher condition number) and the eigenvalues are more irregularly distributed in the upper end of the spectrum, with more scattered values at the top of the interval (see Fig. 12). Accordingly, those systems of equations corresponding to the interpolation function \((1 + r)\) converge more quickly than those obtained from the TPS.
Table 2: Maximum relative errors for the concentration for the interpolation functions \((1 + r)\) and \((r^2 \log r + P_1)\).

<table>
<thead>
<tr>
<th>Mesh</th>
<th>No. of subdomains</th>
<th>Total no. of nodes per subdomain</th>
<th>No. of internal nodes per subdomain</th>
<th>Maximum relative error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mesh I</td>
<td>80</td>
<td>185</td>
<td>1</td>
<td>-200.9 16.3</td>
</tr>
<tr>
<td>Mesh II</td>
<td>257</td>
<td>1–2–4</td>
<td></td>
<td>-145.9 13.6</td>
</tr>
<tr>
<td>Mesh III</td>
<td>297</td>
<td>1–3–5</td>
<td></td>
<td>-149.8 12.9</td>
</tr>
<tr>
<td>Mesh IV</td>
<td>505</td>
<td>5</td>
<td></td>
<td>-123.0 11.0</td>
</tr>
<tr>
<td>Mesh V</td>
<td>200</td>
<td>455</td>
<td>1</td>
<td>-333.6 35.6</td>
</tr>
<tr>
<td>Mesh VI</td>
<td>455</td>
<td>1</td>
<td></td>
<td>-403.9 29.6</td>
</tr>
</tbody>
</table>

Table 3: Efficiency in terms of number of iterations and cpu time per mesh, for the interpolation functions \((1 + r)\) and \((r^2 \log r + P_1)\).

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Matrix</th>
<th>No. of iterations</th>
<th>cpu time (s)</th>
<th>No. of iterations</th>
<th>cpu time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mesh I</td>
<td>(A^T A)</td>
<td>824</td>
<td>1.1678E–01</td>
<td>1031</td>
<td>1.5296E–01</td>
</tr>
<tr>
<td>Mesh II</td>
<td>(A^T A)</td>
<td>1361</td>
<td>2.4046E–01</td>
<td>1802</td>
<td>3.1579E–01</td>
</tr>
<tr>
<td>Mesh III</td>
<td>(A^T A)</td>
<td>1928</td>
<td>3.5987E–01</td>
<td>2787</td>
<td>5.1908E–01</td>
</tr>
<tr>
<td>Mesh IV</td>
<td>(A^T A)</td>
<td>1510</td>
<td>4.0822E–01</td>
<td>2481</td>
<td>6.5132E–01</td>
</tr>
<tr>
<td>Mesh V</td>
<td>(A^T A)</td>
<td>1240</td>
<td>3.7500E–01</td>
<td>1533</td>
<td>4.4441E–01</td>
</tr>
<tr>
<td>Mesh VI</td>
<td>(A^T A)</td>
<td>1275</td>
<td>3.7829E–01</td>
<td>1739</td>
<td>5.0296E–01</td>
</tr>
</tbody>
</table>

Nevertheless, the cpu time remains in the same order of magnitude for the same mesh, and, moreover, the relative error obtained for the concentration through the interpolation function \((r^2 \log r + P_1)\) is one order of magnitude lower than that obtained with \((1 + r)\) (see Fig. 11).

6.2.2 GMRES scheme

As we mentioned in the previous section, solving the normal equation is not a bad option when the eigenvalue distribution agrees fairly well with the singular values of the original system. Thus, we will study the performance of the GMRES algorithm applied to \(A^T A\) for all the six meshes. We will also present the results for two preconditioning techniques, Jacobi and Incomplete LU (ILUT), for accelerating the convergence of the iterative solver. An explicit preconditioner was tested scaling the columns of the matrix in order to make all of them have the same norm. We are not reporting those results, due to a lack of accuracy in the attained solutions.
In Fig. 13 we present the concentration profile per mesh for the interpolation functions \((1 + r)\) in part (a) and \((r^2 \log r + P_1)\) in part (b). In Fig. 14 the corresponding relative errors (%) on the concentration are shown. Since we run the system of equations related to every mesh for different preconditioners, we only plot the results for the most efficient preconditioned matrix. We should point out here, that the accuracy attained for the different preconditioners for a certain mesh is nearly the same, with the exception of mesh V for the augmented TPS interpolation function. We will comment on this matter below.

Again, for this method, the most accurate solution is obtained for mesh IV (with five internal nodes per subdomain) using as before the augmented TPS function.

As regards the relative error for the concentration, we observe that for the function \((1 + r)\) the performance of the GMRES algorithm reaches the same level of accuracy as the CG method (see Figs 13a and 14a). Whereas for the function \((r^2 \log r + P_1)\) we find some small differences in comparison with CG. While the general shape of the error function remains the same, it is possible to observe some irregularities, especially for mesh V (see Fig. 14b). The range of \(x\) coordinates in which we spot those inconsistencies matches the interval of the mesh where the boundary nodes of the subdomains have their maximum proximity, namely for \(0.3m \leq x \leq 0.5m\) (see Fig. 8a). Some irregularities found for meshes II, III and IV can also be related to the closeness of the mesh nodes, this time between internal...
Nodes and boundary nodes, as well as between internal nodes with each other. We have to mention at this point that in the limit of a mesh refinement process, the inconsistencies found (as far as stability is concerned) due to the proximity between boundary nodes outweigh those related to the vicinity of internal nodes, proving GMRES to be very sensitive to the mesh width parameter.

In Fig. 15 we show the differences in the attained accuracy for the concentration, when different preconditioners are applied to the system of equations corresponding to mesh V. This fact highlights the influence of the mesh width parameter in the

---

Figure 13: GMRES semi-log concentration profile per mesh for the interpolation functions (a) \((1 + \frac{1}{r})\) and (b) \((\frac{r^2 \log r}{P_1})\).
convergence of the GMRES algorithm, as it is for mesh V where this parameter takes the smallest values for coordinates $0.3m \leq x \leq 0.5m$.

In Table 4 we report the maximum relative errors (%) of the concentration for the different preconditioning schemes and the two interpolation functions. The irregularities mentioned above for mesh V can also be observed in this table. As mentioned for CG, a lack of convergence is found for the DRM-MD when the approximation power is increased (meshes V and VI).

In Table 5 the number of iterations and $cpu$ time are presented. The numbers in parenthesis in these two tables mean either the relative error was attained in too many iterations or for those number of iterations and $cpu$ time the obtained relative error was not acceptable.
Figure 15: Semi-log concentration profile, mesh V, interpolation function $(r^2 \log r + P_1)$.

Table 4: Maximum relative errors for the concentration for the interpolation functions $(1 + r)$ and $(r^2 \log r + P_1)$.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Without preconditioner</th>
<th>With preconditioner</th>
<th>Without preconditioner</th>
<th>With preconditioner</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$1 + r$</td>
<td>$r^2 \log r + P_1$</td>
<td></td>
</tr>
<tr>
<td>Mesh I</td>
<td>-201.2</td>
<td>-202.1</td>
<td>-200.8</td>
<td>16.8</td>
</tr>
<tr>
<td>Mesh II</td>
<td>-146.1</td>
<td>-146.1</td>
<td>-13.8</td>
<td>13.4</td>
</tr>
<tr>
<td>Mesh III</td>
<td>-150.2</td>
<td>(-149.5)</td>
<td>-151.2</td>
<td>13.0</td>
</tr>
<tr>
<td>Mesh IV</td>
<td>-123.0</td>
<td>-123.5</td>
<td>-124.8</td>
<td>11.0</td>
</tr>
<tr>
<td>Mesh V</td>
<td>-332.4</td>
<td>(-335.4)</td>
<td>49.6</td>
<td>59.5</td>
</tr>
<tr>
<td>Mesh VI</td>
<td>-404.7</td>
<td>(-409.0)</td>
<td>27.4</td>
<td>31.9</td>
</tr>
</tbody>
</table>

From the results in Tables 4 and 5 we observe that the GMRES method presents an erratic behaviour. There is not a single GMRES scheme which will perform better for all the meshes, or for the same interpolation function.

The running time for unpreconditioned GMRES is similar for the two RBFs. The cpu time increases with the total number of nodes per mesh, this increment is more
Table 5: Efficiency in terms of number of iterations and cpu time per mesh for the interpolation functions \((1 + r)\) and \((r^2 \log r + P_1)\).

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Matrix</th>
<th>((1 + r)) iterations</th>
<th>((1 + r)) cpu time (s)</th>
<th>((r^2 \log r + P_1)) iterations</th>
<th>((r^2 \log r + P_1)) cpu time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>(A^TA)</td>
<td>232</td>
<td>2.6053E−01</td>
<td>224</td>
<td>2.4934E−01</td>
</tr>
<tr>
<td></td>
<td>(A^TA) Jacobi preconditioner</td>
<td>76</td>
<td>4.9342E−02</td>
<td>92</td>
<td>5.0658E−02</td>
</tr>
<tr>
<td></td>
<td>(A^TA) ILU preconditioner</td>
<td>12</td>
<td>7.5658E−03</td>
<td>14</td>
<td>1.7763E−02</td>
</tr>
<tr>
<td>II</td>
<td>(A^TA)</td>
<td>310</td>
<td>5.5329E−01</td>
<td>305</td>
<td>5.2928E−01</td>
</tr>
<tr>
<td></td>
<td>(A^TA) Jacobi preconditioner</td>
<td>3500</td>
<td>2.5612E+00</td>
<td>135</td>
<td>1.1974E−01</td>
</tr>
<tr>
<td></td>
<td>(A^TA) ILU preconditioner</td>
<td>–</td>
<td>–</td>
<td>(356)</td>
<td>(1.3230E+00)</td>
</tr>
<tr>
<td>III</td>
<td>(A^TA)</td>
<td>358</td>
<td>7.9605E−01</td>
<td>357</td>
<td>7.8026E−01</td>
</tr>
<tr>
<td></td>
<td>(A^TA) Jacobi preconditioner</td>
<td>–</td>
<td>–</td>
<td>165</td>
<td>1.8651E−01</td>
</tr>
<tr>
<td></td>
<td>(A^TA) ILU preconditioner</td>
<td>209</td>
<td>4.8092E−01</td>
<td>(265)</td>
<td>(6.8980E−01)</td>
</tr>
<tr>
<td>IV</td>
<td>(A^TA)</td>
<td>401</td>
<td>1.5730E+00</td>
<td>453</td>
<td>2.0115E+00</td>
</tr>
<tr>
<td></td>
<td>(A^TA) Jacobi preconditioner</td>
<td>147</td>
<td>2.4836E−01</td>
<td>174</td>
<td>3.3849E−01</td>
</tr>
<tr>
<td></td>
<td>(A^TA) ILU preconditioner</td>
<td>276</td>
<td>9.8026E−01</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>V</td>
<td>(A^TA)</td>
<td>502</td>
<td>2.9967E+00</td>
<td>483</td>
<td>2.7980E+00</td>
</tr>
<tr>
<td></td>
<td>(A^TA) Jacobi preconditioner</td>
<td>–</td>
<td>–</td>
<td>186</td>
<td>5.0461E−01</td>
</tr>
<tr>
<td></td>
<td>(A^TA) ILU preconditioner</td>
<td>–</td>
<td>–</td>
<td>187</td>
<td>7.6414E−01</td>
</tr>
<tr>
<td>VI</td>
<td>(A^TA)</td>
<td>506</td>
<td>3.0444E+00</td>
<td>487</td>
<td>2.8270E+00</td>
</tr>
<tr>
<td></td>
<td>(A^TA) Jacobi preconditioner</td>
<td>–</td>
<td>–</td>
<td>205</td>
<td>5.7039E−01</td>
</tr>
<tr>
<td></td>
<td>(A^TA) ILU preconditioner</td>
<td>21</td>
<td>6.4803E−02</td>
<td>37</td>
<td>7.9276E−02</td>
</tr>
</tbody>
</table>
distinctive when there are more subdomains per mesh (meshes V and VI), rather than more internal nodes per subdomain (meshes I to IV). Preconditioning softens this mesh size dependence, at the expense of losing accuracy.

In an attempt to understand the performance of the incomplete LU preconditioner, we will briefly study the eigenvalue distribution of this preconditioner.

The spectrum of the ILU matrices lies on the complex plane. For those cases in which the distribution of the eigenvalues remains on the positive part of the real axis, and furthermore their distribution takes the shape of an ellipse symmetrically oriented along the real right axis, the incomplete factorization method can be used as a good smoother for multidomain methods, improving the rate of convergence. The incomplete factorization scheme applied to mesh I (Fig. 16) turns out to be the most efficient accelerator, reducing the convergence time by one order of magnitude. When the same factorization is applied to mesh VI (Fig. 17) again the performance of GMRES is improved by dropping the cpu time by one order of magnitude compared to the unpreconditioned results (Table 5). For meshes II to V the incomplete factorization is either a poor option or it is merely not applicable.

![Eigenvalue distribution of the incomplete LU preconditioner](image)

Figure 16: Eigenvalue distribution of the incomplete LU preconditioner for the interpolation functions (a) \((1 + r)\) and (b) \((r^2 \log r + P_1)\) (mesh I).
Figure 17: Eigenvalue distribution of the incomplete LU preconditioner for the interpolation functions (a) $(1 + r)$ and (b) $(r^2 \log r + P_1)$ (mesh VI).

(see Table 5). This is a consequence of the high condition number of the resulting preconditioner (as mentioned in Section 5.4.2 and shown in Table 1), and also to the eigenvalues distributed on both the positive and negative parts of the real axis.

The fact that the numerical results improve when the subdomains have got more number of internal nodes can be related to the intrinsic properties of the system of equations arising from different discretizations of the domain. For this reason we study the distribution of eigenvalues for the six meshes and the two interpolation functions. When using a very simple interpolation function as is $(1 + r)$, the accuracy of the approximated solution improves when we increase the number of internal nodes per subdomain. The gap in the distribution of eigenvalues for mesh I (Fig. 18a) is closed when more internal nodes are considered and this is reflected in a more uniform distribution of the eigenvalues along with many nearly equal singular values clustered around the number 1. On the other hand, when we use $(r^2 \log r + P_1)$ as the interpolation function, the results are more accurate and the increment of the number of internal nodes per subdomain has a less dramatic influence on the improvement of the approximated solution (Fig. 18b): there is not
such a gap in the spectrum of mesh I, still we can notice a more uniform distribution of eigenvalues clustered around the number 1 as the number of internal nodes increases. In Fig. 18 we present the changes in the eigenvalue distribution for the coefficient matrices that arise from DRM-MD when the domain is divided into 80 subdomains, and the number of internal nodes per subdomain is increased from one to five, namely from mesh I to mesh IV, for the two RBFs used.

6.2.3 LSQR scheme

In this section we present the numerical results obtained by the LSQR algorithm applied to the overdetermined system of equations obtained from DRM-MD. To study the performance of this method we solve the tested problem for the six meshes previously considered. Besides, we check the behaviour of three different preconditioners. First we preprocessed the matrix in order to place large entries on the main diagonal, which improves the stability of the method (due to the way in which the assembly of the whole system of equations is done in DRM-MD, we get zero values on the main diagonal). Although this practice may destroy some useful sparsity structure of $A$ (see Fig. 19), it allows us to use the new diagonal as a Jacobi preconditioner. For this purpose we take the diagonal of the square part of

Figure 18: Eigenvalue distribution for meshes I to IV for the interpolation functions (a) $(1 + r)$ and (b) $(r^2 \log r + P_1)$. 
the preprocessed matrix in order to use a diagonal matrix $D(n, n)$. For the diagonal preconditioning we perform $[A(m, n) \cdot D^{-1}(n, n)]$, obtaining the preconditioned matrix with dimensions $(m, n)$.

Column scaling is another preconditioning strategy that we experimented with, so as to obtain an equivalent system with a matrix that will be better conditioned (see Table 6), thereby speeding up the solution process and having a beneficial effect on the accuracy of the computed solution.

Table 6: Condition numbers of the matrix $A$ before and after scaling.

<table>
<thead>
<tr>
<th>Mesh – $A(m, n)$ – RBF</th>
<th>Condition number $A$</th>
<th>$A$-scaled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mesh I (400, 343)(1 + $r$)</td>
<td>5.7377E+02</td>
<td>3.2280E+01</td>
</tr>
<tr>
<td>Mesh I (400, 343)($r^2 \log r + p$)</td>
<td>8.3479E+02</td>
<td>3.8935E+01</td>
</tr>
<tr>
<td>Mesh II (472, 415)(1 + $r$)</td>
<td>8.4964E+02</td>
<td>4.4604E+01</td>
</tr>
<tr>
<td>Mesh II (472, 415)($r^2 \log r + p$)</td>
<td>1.2753E+03</td>
<td>5.8059E+01</td>
</tr>
<tr>
<td>Mesh III (512, 454)(1 + $r$)</td>
<td>9.1379E+02</td>
<td>5.3534E+01</td>
</tr>
<tr>
<td>Mesh III (512, 454)($r^2 \log r + p$)</td>
<td>1.5056E+03</td>
<td>8.4657E+01</td>
</tr>
<tr>
<td>Mesh IV (720, 663)(1 + $r$)</td>
<td>8.4240E+02</td>
<td>6.2815E+01</td>
</tr>
<tr>
<td>Mesh IV (720, 663)($r^2 \log r + p$)</td>
<td>2.1275E+03</td>
<td>7.8179E+01</td>
</tr>
<tr>
<td>Mesh V (1000, 853)(1 + $r$)</td>
<td>1.1874E+03</td>
<td>9.6298E+01</td>
</tr>
<tr>
<td>Mesh V (1000, 853)($r^2 \log r + p$)</td>
<td>1.5605E+03</td>
<td>1.3422E+02</td>
</tr>
<tr>
<td>Mesh VI (1000, 853)(1 + $r$)</td>
<td>6.6428E+02</td>
<td>7.6106E+01</td>
</tr>
<tr>
<td>Mesh VI (1000, 853)($r^2 \log r + p$)</td>
<td>1.0077E+03</td>
<td>1.0086E+02</td>
</tr>
</tbody>
</table>

Figure 19: Matrix $A$ (a) before and (b) after preprocessing.
Figure 20: LSQR semi-log concentration profile per mesh for the interpolation functions (a) $(1 + r)$ and (b) $(r^2 \log r + P_1)$.

The results for the concentration of the test problem are reported in Fig. 20, for the six meshes and for the two interpolation functions. Again here we are plotting the most efficient solution which turned out to be, for all the cases, the scaling scheme. When we observe the distribution of the relative errors, in Fig. 21, we can see that no irregularities are present for either the six meshes or the two interpolation functions, as was the case for the numerical results we got for the
CG and GMRES methods (Figs 11 and 14). Moreover, the LSQR algorithm performs better for the same preconditioned scheme, regardless of the mesh tested or the interpolation function used. In Table 7 we report the maximum relative errors (%) for the concentration for the two interpolation functions. We only show the numbers for the scaling scheme because all the other schemes reached the same level of accuracy, the difference between them being in the solving time. As in the
Table 7: Maximum relative errors for the concentration for the interpolation functions \((1 + r)\) and \((r^2 \log r + P_1)\).

<table>
<thead>
<tr>
<th>Mesh</th>
<th>No. of subdomains</th>
<th>Total no. of nodes</th>
<th>No. of internal nodes per subdomain</th>
<th>Maximum relative error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mesh I</td>
<td>80</td>
<td>185</td>
<td>1</td>
<td>-200.9, 16.3</td>
</tr>
<tr>
<td>Mesh II</td>
<td>257</td>
<td>1–2–4</td>
<td>-145.9, 13.6</td>
<td></td>
</tr>
<tr>
<td>Mesh III</td>
<td>297</td>
<td>1–3–5</td>
<td>-149.8, 12.9</td>
<td></td>
</tr>
<tr>
<td>Mesh IV</td>
<td>505</td>
<td>5</td>
<td>-123.0, 11.0</td>
<td></td>
</tr>
<tr>
<td>Mesh V</td>
<td>200</td>
<td>455</td>
<td>1</td>
<td>-333.6, 35.6</td>
</tr>
<tr>
<td>Mesh VI</td>
<td>455</td>
<td>1</td>
<td>-403.9, 29.6</td>
<td></td>
</tr>
</tbody>
</table>

Previous cases the best result is found with the mesh IV and the use of the ATPS function.

CPU time results are presented in Table 8, as well as the number of iterations to converge. It is clear from these results that scaling the matrix is the most reliable preconditioned scheme, which greatly enhances the efficiency of the LSQR solver. Even when the cpu time, reported for a specific mesh and interpolation function, remains at the same order of magnitude as the most efficient implementations of the CG and GMRES methods (Tables 3 and 5), the LSQR convergence method is always more stable than the others, yielding to a more uniform distribution of the relative error.

In Fig. 22 we present the changes in the distribution of the singular values of the coefficient matrices when the number of internal nodes per subdomain is increased from one to five, i.e. for meshes I to IV. Figure 22a corresponds to the RBF \((1 + r)\) and Fig. 22b to the ATPS. As in the case of the distribution of eigenvalues (Section 6.2.2, Fig. 18), the systems of equations with more nearly equal singular values, clustered around 1, attain better solutions.

It is worth, at this point, to take an insight into the distribution of the singular values of the original matrices and their scaled versions. In Figs 23–25 we observe that the singular values of the scaled matrices are in all cases distributed in the interval \((0, 3)\); besides, the curves for both \((1 + r)\) and \((r^2 \log r + P_1)\) have nearly the same distribution. As a consequence of this we find a very regular behaviour of the scaled LSQR algorithm, not showing the high sensitivity to the mesh width parameter we reported for GMRES.

Scaling also makes the matrix norm smaller, resulting in a good attempt to reduce the inaccuracy contributed by roundoff errors.

Up to now we have reported the computed solutions for the concentration. In Figs 26 and 27 we show the numerical results obtained for the derivatives in the \(x\) direction for the ATPS interpolation function. We should recall that Meshes I to IV have the same grid with different distributions of internal points per subdomain.
Iterative Schemes for Solution of Linear Systems Using DRM-MD

Table 8: Efficiency in terms of number of iterations and cpu time per mesh, for the interpolation functions \((1 + r)\) and \((r^2 \log r + P_1)\).

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Matrix</th>
<th>((1 + r))</th>
<th>((r^2 \log r + P_1))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No. of iterations</td>
<td>cpu time (s)</td>
<td>No. of iterations</td>
</tr>
<tr>
<td>I</td>
<td>A</td>
<td>1162</td>
<td>3.4507E−01</td>
</tr>
<tr>
<td></td>
<td>A preprocessor</td>
<td>1159</td>
<td>3.4572E−01</td>
</tr>
<tr>
<td></td>
<td>A Jacobi preconditioner</td>
<td>624</td>
<td>1.7961E−01</td>
</tr>
<tr>
<td></td>
<td>A scale</td>
<td>100</td>
<td>3.1250E−02</td>
</tr>
<tr>
<td></td>
<td>A preprocessor</td>
<td>1776</td>
<td>6.3684E−01</td>
</tr>
<tr>
<td></td>
<td>A Jacobi preconditioner</td>
<td>1766</td>
<td>6.3059E−01</td>
</tr>
<tr>
<td></td>
<td>A scale</td>
<td>860</td>
<td>3.0855E−01</td>
</tr>
<tr>
<td></td>
<td>A scale</td>
<td>120</td>
<td>4.4408E−02</td>
</tr>
<tr>
<td>II</td>
<td>A</td>
<td>3982</td>
<td>1.5681E+00</td>
</tr>
<tr>
<td></td>
<td>A preprocessor</td>
<td>3255</td>
<td>1.2924E+00</td>
</tr>
<tr>
<td></td>
<td>A Jacobi preconditioner</td>
<td>1761</td>
<td>7.2237E−01</td>
</tr>
<tr>
<td></td>
<td>A scale</td>
<td>413</td>
<td>1.6579E−01</td>
</tr>
<tr>
<td>III</td>
<td>A</td>
<td>2131</td>
<td>1.2533E+00</td>
</tr>
<tr>
<td></td>
<td>A preprocessor</td>
<td>2143</td>
<td>1.2543E+00</td>
</tr>
<tr>
<td></td>
<td>A scale</td>
<td>178</td>
<td>1.0757E−01</td>
</tr>
<tr>
<td>IV</td>
<td>A</td>
<td>2595</td>
<td>1.9059E+00</td>
</tr>
<tr>
<td></td>
<td>A preprocessor</td>
<td>2591</td>
<td>1.9224E+00</td>
</tr>
<tr>
<td></td>
<td>A Jacobi preconditioner</td>
<td>1871</td>
<td>1.3997E+00</td>
</tr>
<tr>
<td></td>
<td>A scale</td>
<td>245</td>
<td>1.8421E−01</td>
</tr>
<tr>
<td>V</td>
<td>A</td>
<td>2067</td>
<td>1.5171E+00</td>
</tr>
<tr>
<td></td>
<td>A preprocessor</td>
<td>2070</td>
<td>1.5243E+00</td>
</tr>
<tr>
<td></td>
<td>A Jacobi preconditioner</td>
<td>1122</td>
<td>8.3026E−01</td>
</tr>
<tr>
<td></td>
<td>A scale</td>
<td>249</td>
<td>1.8388E−01</td>
</tr>
</tbody>
</table>

While the three iterative algorithms performed similarly, as far as accuracy in the computed concentration is concerned, we find LSQR to be more reliable than the others for the results on the derivatives, reaching solutions with smaller relative errors.

When we focus on the computed results for the \(x\)-derivatives for meshes I, V and VI – whose common pattern is that they all have one internal point per subdomain, while there is a refinement in the mesh width parameter – we still find that the DRM-MD approach loses convergence with the proximity of the boundary nodes.
Figure 22: Singular value distribution for meshes I to IV: (a) \((1 + r)\) and (b) \((r^2 \log r + P_1)\).

Figure 23: Singular value distribution of the matrices \(A\) and \(A\)-scaled, for the interpolation functions \((1 + r)\) and \((r^2 \log r + P_1)\): (a) mesh I and (b) mesh II.
Iterative Schemes for Solution of Linear Systems Using DRM-MD

Figure 24: Singular value distribution of the matrices $A$ and $A$-scaled, for the interpolation functions $(1 + r)$ and $(r^2 \log r + P_1)$: (a) mesh III and (b) mesh IV.

Figure 25: Singular value distribution of the matrices $A$ and $A$-scaled, for the interpolation functions $(1 + r)$ and $(r^2 \log r + P_1)$: (a) mesh V and (b) mesh VI.
Despite this fact, we observe that the LSQR algorithm attains more stable results (see Figs 28 and 29).

6.3 Direct methods

The efficient solution of a linear system is largely a function of the proper choice of the iterative method. A direct method may sometimes be preferable to an iterative
method. The performance of direct methods, both for dense and sparse systems, is to a great extent that of the factorization of the matrix. This operation is absent in iterative methods (although preconditioners may require a setup phase), and with it, iterative methods do not require dense matrix suboperations, which involve high computational cost. Furthermore, the basic operations in iterative methods often use indirect addressing, depending on the data structure, having a low cost of execution. However, this does not imply anything about the total solution time for a given system.

On the other hand, iterative methods will always retain advantages for certain applications in which we can predict a quick convergence, and they will also have lower storage requirements than direct methods. In spite of this, it is still worth analyzing the performance of some direct methods, in particular to compare their results with indirect solutions.

When we solve an overdetermined system of equations in the least squares sense it always leads us to matrices of the form $A^T A$, which is then needed to be inverted. If we are ready to use $A^T A$, this matrix being symmetric and positive definite, then it has a special and efficient triangular decomposition. Thus, Cholesky decomposition is a useful tool. When we can use it, this decomposition is about a factor of two faster than alternative direct methods for solving linear equations. It is extremely stable numerically and once the matrix is decomposed, the triangular factor can be used to solve the linear system by back substitution.

SVD is amongst a set of very powerful techniques for dealing with sets of equations or matrices that are either singular or else numerically very close to singular. SVD is also the method of choice for solving most linear least-squares problems.

Figure 27: Relative errors for the longitudinal derivative for meshes with 80 subdomains obtained with LSQR. Interpolation function ($r^2 \log r + P_1$).
We are not presenting results for this method as its time consumption is highly uncompetitive.

Golub’s method is a direct method that works directly with $A$ and has the advantage of being considerably more accurate than methods that invert $A^T A$. It is reckoned that about twice as much precision is required to invert $A^T A$ than the
one needed to deal directly with $A$. Golub’s method uses Householder transformations to convert any general overdetermined matrix into an upper triangular form [30].

In this section we describe the results for Golub’s method. The solutions reached by this direct method are in good agreement with those obtained with the indirect methods. The estimated solutions for the concentration and for the derivatives attain the same results as those obtained by the LSQR algorithm. In particular, for mesh IV and the ATPS function (where the lowest relative errors where obtained for the iterative techniques) both direct and indirect techniques yielded the same relative error, the direct solutions being more time-consuming.

In Table 9 the solving time for different preconditioning schemes is shown, where it can be observed that preconditioning has no effect upon the computed solutions for direct solvers. The preconditioning schemes are the same as those applied for the LSQR method.

A comparison of the CPU times for all the implemented iterative techniques and for Golub’s method is presented in Table 10.

7 Conclusions

This work presents a thorough analysis of the performance and reliability of several preconditioned iterative techniques for the solution of linear systems arising from the DRM-MD integral equation approach. In the experiments shown in this work the shape functions used for every boundary element for the approximation of the
Table 9: Efficiency for different preconditioners per mesh for Golub’s method. Interpolation functions \((1 + r)\) and \((r^2 \log r + P_1)\).

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Matrix</th>
<th>((1 + r))</th>
<th>((r^2 \log r + P_1))</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>A</td>
<td>4.4967E−01</td>
<td>4.4901E−01</td>
</tr>
<tr>
<td></td>
<td>A preprocessor</td>
<td>4.5132E−01</td>
<td>4.4934E−01</td>
</tr>
<tr>
<td></td>
<td>A Jacobi</td>
<td>4.5921E−01</td>
<td>4.5757E−01</td>
</tr>
<tr>
<td></td>
<td>A scale</td>
<td>4.6776E−01</td>
<td>4.6678E−01</td>
</tr>
<tr>
<td>II</td>
<td>A</td>
<td>7.6579E−01</td>
<td>7.6645E−01</td>
</tr>
<tr>
<td></td>
<td>A preprocessor</td>
<td>7.6809E−01</td>
<td>7.6612E−01</td>
</tr>
<tr>
<td></td>
<td>A Jacobi</td>
<td>7.8059E−01</td>
<td>7.7928E−01</td>
</tr>
<tr>
<td></td>
<td>A scale</td>
<td>7.9145E−01</td>
<td>7.9112E−01</td>
</tr>
<tr>
<td>III</td>
<td>A</td>
<td>9.8243E−01</td>
<td>9.9178E−01</td>
</tr>
<tr>
<td></td>
<td>A preprocessor</td>
<td>9.7826E−01</td>
<td>9.9638E−01</td>
</tr>
<tr>
<td></td>
<td>A Jacobi</td>
<td>1.0172E+00</td>
<td>1.0388E+00</td>
</tr>
<tr>
<td></td>
<td>A scale</td>
<td>1.0112E+00</td>
<td>1.0211E+00</td>
</tr>
<tr>
<td>IV</td>
<td>A</td>
<td>2.9707E+00</td>
<td>2.9655E+00</td>
</tr>
<tr>
<td></td>
<td>A preprocessor</td>
<td>3.0257E+00</td>
<td>2.9674E+00</td>
</tr>
<tr>
<td></td>
<td>A Jacobi</td>
<td>3.0158E+00</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>A scale</td>
<td>3.0822E+00</td>
<td>3.0319E+00</td>
</tr>
<tr>
<td>V</td>
<td>A</td>
<td>7.2030E+00</td>
<td>7.2049E+00</td>
</tr>
<tr>
<td></td>
<td>A preprocessor</td>
<td>7.3148E+00</td>
<td>7.3717E+00</td>
</tr>
<tr>
<td></td>
<td>A Jacobi</td>
<td>7.4895E+00</td>
<td>7.9793E+00</td>
</tr>
<tr>
<td></td>
<td>A scale</td>
<td>7.3217E+00</td>
<td>7.3168E+00</td>
</tr>
<tr>
<td>VI</td>
<td>A</td>
<td>7.3095E+00</td>
<td>7.1974E+00</td>
</tr>
<tr>
<td></td>
<td>A preprocessor</td>
<td>7.3155E+00</td>
<td>7.1970E+00</td>
</tr>
<tr>
<td></td>
<td>A Jacobi</td>
<td>7.8112E+00</td>
<td>7.9428E+00</td>
</tr>
<tr>
<td></td>
<td>A scale</td>
<td>7.4164E+00</td>
<td>7.3151E+00</td>
</tr>
</tbody>
</table>

geometry and for the considered governing equations are linear functions. For the interpolation functions used by the DRM we test the results for two different radial basis interpolation functions, namely \((1 + r)\) and \((r^2 \log r + P_1)\). Besides, we run all these series of tests for six different meshes, in which the whole domain is subdivided, refining both the grid and the number of internal nodes per subdomain. This allows us to understand the convergence behaviour of the DRM-MD. From the analysis per method with different preconditioners we observed that the most accurate situation was consistently obtained for \((r^2 \log r + P_1)\). The most accurate results for a single mesh and different preconditoning schemes turned out to be also the most efficient.
Table 10: Efficiency per method with the best preconditioner for every mesh. Interpolation functions \((1 + r)\) and \((r^2 \log r + P_1)\).

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Method</th>
<th>((1 + r)) cpu time (s)</th>
<th>((r^2 \log r + P_1)) cpu time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Golub</td>
<td>4.4967E−01</td>
<td>4.4901E−01</td>
</tr>
<tr>
<td></td>
<td>Scaled CGN</td>
<td>2.6217E−01</td>
<td>2.0526E−01</td>
</tr>
<tr>
<td></td>
<td>CG</td>
<td>1.1678E−01</td>
<td>1.5296E−01</td>
</tr>
<tr>
<td></td>
<td>ILU GMRES</td>
<td>7.5658E−03</td>
<td>1.7763E−02</td>
</tr>
<tr>
<td></td>
<td>Scaled LSQR</td>
<td>3.1250E−02</td>
<td>5.9539E−02</td>
</tr>
<tr>
<td></td>
<td>Scaled CGN</td>
<td>6.5000E−01</td>
<td>8.3355E−01</td>
</tr>
<tr>
<td></td>
<td>CG</td>
<td>2.4046E−01</td>
<td>3.1579E−01</td>
</tr>
<tr>
<td></td>
<td>ILU GMRES</td>
<td>1.6875E−01</td>
<td>5.2928E−01</td>
</tr>
<tr>
<td></td>
<td>Scaled LSQR</td>
<td>4.4408E−02</td>
<td>1.0954E−01</td>
</tr>
<tr>
<td>II</td>
<td>Golub</td>
<td>7.6579E−01</td>
<td>7.6612E−01</td>
</tr>
<tr>
<td></td>
<td>Scaled CGN</td>
<td>6.5000E−01</td>
<td>8.3355E−01</td>
</tr>
<tr>
<td></td>
<td>CG</td>
<td>2.4046E−01</td>
<td>3.1579E−01</td>
</tr>
<tr>
<td></td>
<td>ILU GMRES</td>
<td>1.6875E−01</td>
<td>5.2928E−01</td>
</tr>
<tr>
<td></td>
<td>Scaled LSQR</td>
<td>4.4408E−02</td>
<td>1.0954E−01</td>
</tr>
<tr>
<td>III</td>
<td>Golub</td>
<td>9.8243E−01</td>
<td>9.9178E−01</td>
</tr>
<tr>
<td></td>
<td>Scaled CGN</td>
<td>1.0882E+00</td>
<td>1.5582E+00</td>
</tr>
<tr>
<td></td>
<td>CG</td>
<td>3.5987E−01</td>
<td>5.1908E−01</td>
</tr>
<tr>
<td></td>
<td>ILU GMRES</td>
<td>4.8092E−01</td>
<td>1.8651E−01</td>
</tr>
<tr>
<td></td>
<td>Scaled LSQR</td>
<td>1.6579E−01</td>
<td>2.1579E−01</td>
</tr>
<tr>
<td>IV</td>
<td>Golub</td>
<td>3.2375E+00</td>
<td>3.1941E+00</td>
</tr>
<tr>
<td></td>
<td>Scaled CGN</td>
<td>2.0766E+00</td>
<td>1.8967E+00</td>
</tr>
<tr>
<td></td>
<td>CG</td>
<td>4.0822E−01</td>
<td>6.5132E−01</td>
</tr>
<tr>
<td></td>
<td>Jacobi GMRES</td>
<td>2.4836E−01</td>
<td>3.3849E−01</td>
</tr>
<tr>
<td></td>
<td>Scaled LSQR</td>
<td>1.0625E−01</td>
<td>2.1131E−01</td>
</tr>
<tr>
<td>V</td>
<td>Golub</td>
<td>7.2030E+00</td>
<td>7.2049E+00</td>
</tr>
<tr>
<td></td>
<td>Scaled CGN</td>
<td>1.8171E+00</td>
<td>1.4372E+00</td>
</tr>
<tr>
<td></td>
<td>CG</td>
<td>3.7500E−01</td>
<td>4.4441E−01</td>
</tr>
<tr>
<td></td>
<td>GMRES</td>
<td>2.9967E+00</td>
<td>5.0461E−01</td>
</tr>
<tr>
<td></td>
<td>Scaled LSQR</td>
<td>1.8421E−01</td>
<td>3.1612E−01</td>
</tr>
<tr>
<td>VI</td>
<td>Golub</td>
<td>7.3095E+00</td>
<td>7.1970E+00</td>
</tr>
<tr>
<td></td>
<td>Scaled CGN</td>
<td>2.8345E+00</td>
<td>2.0191E+00</td>
</tr>
<tr>
<td></td>
<td>CG</td>
<td>3.7829E−01</td>
<td>5.0296E−01</td>
</tr>
<tr>
<td></td>
<td>ILU GMRES</td>
<td>6.4803E−02</td>
<td>7.9276E−02</td>
</tr>
<tr>
<td></td>
<td>Scaled LSQR</td>
<td>1.8388E−01</td>
<td>2.7664E−01</td>
</tr>
</tbody>
</table>

In relation to the CG and GMRES Krylov subspace methods, we solved the explicit normal system of equations after having found a good agreement between the singular eigenvalue distributions of \(A\) and \(A^T A\). GMRES outperforms CG, but proves to be too sensitive to the mesh width parameter, resulting in an erratic behaviour of the method. With reference to preconditioning schemes, GMRES improves its convergence rate through different preconditioners, although the
accelerator happened to be different for every test carried out, showing irregularities in its performance once more.

LSQR yields the most accurate and efficient solutions when used in combination with a scaling of the input matrix. This not only has a stabilizing effect on the computed solutions but also alleviates the calculations of roundoff errors.

As we can notice from the numerical results, not every method will work on every problem type, and knowledge of matrix properties is the main criterion for selecting an iterative technique. For this reason a study on the intrinsic properties of the systems is presented. A concluding remark on this subject is the regularization on the distribution of the singular (eigen) values of the matrices when the number of the internal nodes per subdomain is increased.

With reference to the convergence of the DRM-MD, it is clear that the computed solutions improve when we increase the number of internal nodes per subdomain for a single grid in which the domain is divided. On the other hand, when the refinement of the mesh consists in increasing the number of subdomains, the solution deteriorates losing accuracy, and also efficiency due to a higher storage requirement. We should point out here that the proximity of interior nodes also generates a reduction in accuracy, but this is a local problem and far less important than what is observed due to the proximity of boundary nodes. Therefore, it might be worth seeking an adaptive meshing technique capable of considering the increment of internal nodes per subdomain prior to the refining of the mesh itself.

References


Domain Decomposition Techniques for Boundary Elements


