Induced Dimension Reduction method for solving linear matrix equations

R. Astudillo and M. B. van Gijzen
Induced Dimension Reduction method for solving linear matrix equations

Reinaldo Astudillo and Martin B. van Gijzen

Delft University of Technology, Delft Institute of Applied Mathematics, The Netherlands

Abstract

This paper discusses the solution of large-scale linear matrix equations using the Induced Dimension Reduction method (IDR(s)). IDR(s) was originally presented to solve system of linear equations, and is based on the IDR(s) theorem. We generalize the IDR(s) theorem to solve linear problems in any finite-dimensional space. This generalization allows us to develop IDR(s) algorithms to approximate the solution of linear matrix equations. The IDR(s) method presented here has two main advantages; firstly, it does not require the computation of inverses of any matrix, and secondly, it allows incorporation of preconditioners. Additionally, we present a simple preconditioner to solve the Sylvester equation based on a fixed point iteration. Several numerical examples illustrate the performance of IDR(s) for solving linear matrix equations. We also present the software implementation.

Keywords: Matrix linear equations, Krylov subspace methods, Induced Dimension Reduction method, Preconditioner, Numerical software.

1 Introduction

In this work we extended the Induced Reduction Dimension method (IDR(s) [1]) to approximate the solution of linear matrix equations,

\[ \sum_{j=1}^{k} A_j X B_j^T = C, \]

where the \( A_1, A_2, \ldots, A_k \) are in \( \mathbb{C}^{n \times n} \), \( B_1, B_2, \ldots, B_k \) are in \( \mathbb{C}^{m \times m} \), \( C \in \mathbb{C}^{n \times m} \), and \( X \in \mathbb{C}^{n \times m} \) is unknown. Solving equation (1) is equivalent to solve a linear system of equations. Defining \( \text{vec}(X) \) as the vector of order \( n \times m \) created by stacking the columns of the matrix \( X \), we can write (1) as,

\[ \left( \sum_{j=1}^{k} B_k \otimes A_k \right) \text{vec}(X) = \text{vec}(C). \]
Throughout this document, we only consider the case when the coefficient matrix of the system of linear equations (2) is non-singular, i.e., Eq. (1) has guaranteed the existence and uniqueness of their solution. For the general case of (1), the conditions to ensure existence and uniqueness of its solution are not fully established. However, in the cases of the Sylvester and Lyapunov equation, the conditions for existence and uniqueness of their solution are known. For the Sylvester equation,
\[ AX + XB = C, \] (3)
the condition for the existence and uniqueness of the solution is that the matrices $A$ and $-B$ do not have any common eigenvalue. The Lyapunov equation,
\[ AX + XA^T = C, \] (4)
has a unique solution when the eigenvalues of $A$ hold that $\lambda_i + \lambda_j \neq 0$ for $1 \leq i, j \leq n$.

Linear matrix equations appear in different areas such as complex networks, and system and control theory (see [2] and its references). Another important source of linear matrix equations is the numerical solution of differential equations. Discretization of differential equations lead to linear systems or parametrized linear systems, and in some cases, they can be rewritten as a Sylvester equations. In this work, we emphasize this kind of examples. We present numerical tests of Sylvester equations originated from the discretization of time-dependant linear systems, and convection-diffusion equations.

Methods to solve (1) have mostly focused on the solution of the particular cases of the Lyapunov and Sylvester equation. For small and dense matrices, one of the earliest algorithms to solve Sylvester equation (7) was proposed by Bartels and Stewart [3]. This algorithm relies on the computation of the Schur decomposition of the matrices $A$ and $B$, and then solve a block upper triangular matrix. An improvement of the Bartels-Stewart method was introduced by Golub, Nash and Van Loan in [4] using the Hessenberg factorization of matrices $A$ and $B$.

Solving large scale linear matrix equation is an active research area. The Alternating Direction Implicit (ADI) method, proposed by Peaceman and Rachford [5], has been adapted to solve the Sylvester equation by Ellner and Wachspress in [6]. The ADI method has been widely used for solving matrix equations, for example, Benner, Li, and Truhar extended the ADI method for Low-rank requirements in [7]. However, the parameter selection in ADI is not trivial and its performance strongly depends on it (see for example [8], and [9]).

Krylov subspace methods have also been applied to solve matrix equations. Saad in [10] proposed to solve low-rank Lyapunov equation [1] using a projection over Krylov subspace
\[ K_m(A, x) = \text{span}\{x, Ax, \ldots, A^{m-1}x\}. \]
To improve the speed of these projection method, based of the work Druskin and Knizhnerman [11], Simoncini proposed the use of the extended Krylov subspace
\[ EK_m(A, X) = K_m(A, X) + K_m(A^{-1}, A^{-1}X), \]
for solving the low-rank Lyapunov equation [12]. For the case of the Sylvester equation, the extended Krylov subspace projection method was used by Druskin and Simoncini in [13]. The main disadvantages of those methods is the use of the inverse of the matrices involved or its factorization, which might be prohibitive for large and unstructured matrices.

Another approach to solve linear matrix equations based on Krylov methods, is to consider the relation between a linear matrix equations and solving linear systems. Hochbruck and Starke applied the Quasi-Minimal Residual method (QMR) [14] to solve the system of
linear equations obtained from the Lyapunov equation. A similar idea was proposed by Jbilou, Messaoudi, and Sadok in [15] to solve Lyapunov and Sylvester equations using block versions of the Full Orthogonalization method (FOM) [16] and the Generalized Minimal Residual method (GMRES) [17] called Global variants. This approach does not need to calculate any inverse of a matrix, however, it can suffer from slow convergence, in which case, it is necessary to apply preconditioners. For a more detailed description of the state-of-art of matrix equation solvers see [2].

In this work, we propose a variant of Induced Dimension Reduction method (IDR(s)) for solving linear matrix equations. IDR(s) was originally proposed in [1] as an iterative method to solve large, sparse and non-symmetric system of linear equations

\[ Ax = b, \]

where \( A \in \mathbb{C}^{n \times n} \) is the coefficient matrix, \( b \) is the right-hand side vector in \( \mathbb{C}^{n} \), and \( x \in \mathbb{C}^{n} \) is unknown. IDR(s) has been adapted to solve other related problems like solving block linear systems [18], multi-shift linear systems [19, 20], and eigenvalue problems [21, 22]. IDR(s) is based on the IDR(s) theorem. In this paper, we generalize the IDR(s) theorem to solve linear problems in any finite-dimensional space. Using this generalization, we develop an IDR(s) algorithm to approximate the solution of linear matrix equations.

1.1 Notation

We use the following notation: column vectors are represented by bold-face, lower case letters and capital letters denote matrices. For a matrix \( A \), \( A^T \) represents its transpose and the \( i \)-th column is denoted by \( A_i \). Greek lower case letters represent scalars. \( \| \cdot \| \) represents the classical Euclidean norm, and \( \| \cdot \|_F \) is the Frobenius norm induced by the Frobenius inner product \( \langle A, B \rangle_F = \text{trace}(A^T B) \). Subspaces are denoted by uppercase calligraphic letters with the exception of \( \mathcal{A}, \mathcal{I}, \) and \( \mathcal{M} \) that represent linear operators. \( I_n \) is the identity matrix of order \( n \), and wherever the context is clear the subindex \( n \) is eliminated.

2 IDR(s) for linear operators

This section extends the IDR(s) method to solve linear matrix equations of the form [1]. We present an alternative form of the IDR(s) theorem. First, we would like to draw the attention of the reader to the proof of Theorem 2.1 in [1]. In this proof, the authors only use that \( \mathbb{C}^n \) is a linear subspace, and that \( A \) is a linear operator on this linear subspace. Using these facts, we can generalize the IDR(s) theorem to any finite-dimensional linear subspace \( D \) with \( A \) as linear operator defined on the same linear subspace. Corollary [14] summarizes this result.

**Corollary 1.1.** Let \( A \) be any linear operator over a finite dimensional subspace \( D \) and \( \mathcal{I} \) the identity operator over the same subspace. Let \( S \) be any (proper) subspace of \( D \). Define \( D_0 \triangleq D \), if \( S \) and \( D_0 \) do not share a nontrivial invariant subspace of the operator \( A \), then the sequence of subspace \( D_j \), defined as

\[ D_j \equiv (\mathcal{I} - \omega_j A)(D_{j-1} \cap S) \quad j = 0, 1, 2 \ldots, \]

with \( \omega_j \)'s nonzero scalars, have the following properties,

1. \( D_{j+1} \subset D_j \), for \( j \geq 0 \) and
2. \( \text{dimension}(\mathcal{G}_{j+1}) < \text{dimension}(\mathcal{G}_j) \) unless \( \mathcal{G}_j = \{0\} \).

**Proof.** The proof is analogous to the one presented in [1].

In [13], Du et al. present another generalization of the original IDR(s) theorem, this generalization is used to derive an IDR(s) method for solving block systems of linear equations. Corollary [1] has a broader scope; we apply this corollary to solve different types of linear matrix equations. Before presenting examples of IDR(s) for solving matrix equations, we show an application of the IDR theorem in the linear space of the polynomials functions, the purpose of this example is to show the flexibility of Corollary [1].

**Example 0:** (Finding a polynomial function such that \( u'' = 12x \)) We are interested in find one solution of the problem, \( u'' = 12x \).

We apply IDR(1) over the subspace \( \mathcal{D} = \{ p : p(x) = \sum_{i=0}^{5} \alpha_i x^i, \text{with} \, \alpha_i \in \mathbb{R} \} \) with inner product \( \langle f, g \rangle = \int_0^1 f(x)g(x)dx \), \( \mathcal{S} = \{ p \in \mathcal{D} : \langle p, P \rangle = 0, \text{with} \, P = 1 \} \), and the operators \( \mathcal{I}, \mathcal{A} : \mathcal{D} \rightarrow \mathcal{D} \) are defined as \( \mathcal{I}(g) = g \) and \( \mathcal{A}(g) = g'' \). The procedure applied in this example was originally described in [1].

- **Subspace \( \mathcal{G}_0 \equiv \mathcal{D} \) (Initial guesses given in \( \mathcal{G}_0 \)):**

\[
\begin{align*}
  u_0 &= x^5, & r_0 &= -20x^3 + 12x \\
  u_1 &= x & r_1 &= 12x
\end{align*}
\]

- **Subspace \( \mathcal{G}_1 \equiv (\mathcal{I} - \omega_1 \mathcal{A})(\mathcal{G}_0 \cap \mathcal{S}) \) (with \( \omega_1 = 1 \)):**

\[
\begin{align*}
  \beta_1 &= \frac{\langle P, r_1 \rangle}{\langle P, r_1 - r_0 \rangle} = \frac{6}{5} \\
  v_1 &= r_1 - \beta_1 (r_1 - r_0) = -24x^3 + 12x \quad \text{in} \, \mathcal{G}_0 \cap \mathcal{S} \\
  r_2 &= (\mathcal{I} - \omega_1 \mathcal{A})v_1 = -24x^3 + 156x \quad \text{1st vector in} \, \mathcal{G}_1 \\
  u_2 &= u_1 + \omega_1 v_1 - \beta_1 (u_1 - u_0) = \frac{6x^5}{5} - 24x^3 + \frac{59x}{5} \\
  \beta_2 &= \frac{\langle P, r_2 \rangle}{\langle P, r_2 - r_1 \rangle} = \frac{12}{11} \\
  v_2 &= r_2 - \beta_2 (r_2 - r_1) = \frac{24x^3}{11} - \frac{12x}{11} \quad \text{in} \, \mathcal{G}_0 \cap \mathcal{S} \\
  r_3 &= (\mathcal{I} - \omega_1 \mathcal{A})v_2 = \frac{24x^3}{11} - \frac{156x}{11} \quad \text{2nd vector in} \, \mathcal{G}_1 \\
  u_3 &= u_2 + \omega_1 v_2 - \beta_2 (u_2 - u_1) = -\frac{6x^5}{55} + \frac{48x^3}{11} - \frac{59x}{55}
\end{align*}
\]
Subspace $G_2 \equiv (I - \omega_2A)(G_1 \cap S)$ (with $\omega_2 = 1$):

$$
\beta_3 = \frac{\langle P, r_2 \rangle}{\langle P, r_2 - r_1 \rangle} = \frac{1}{12}
$$

$$
v_3 = r_3 - \beta_2(r_3 - r_2) = 0 \quad \text{in } (G_1 \cap S)
$$

$$
r_4 = (I - \omega_1A)v_3 = 0 \quad \text{in } G_2
$$

$$
u_4 = u_3 + \omega_2v_3 - \beta_3(u_3 - u_2) = 2x^3
$$

As in [23], we rewrite problems [1] as

$$
A(X) = C, \quad (6)
$$

where $A(X) = \sum_{j=1}^{k} A_jXB_j$. Using Corollary 1.1, we are able to create residuals $R_k = C - A(X_k)$ of the problem [6] in the shrinking and nested subspaces $G_j$ and obtain the approximations $X_k$. Only changing the definition of the operator $A$ and the subspace $D$, we are able to approximate the solution of the linear matrix equation using IDR(s). Assuming that the space $S$ is the null space of the set $P = \{P_1, P_2, \ldots, P_s\}$, and the approximations $\{X_i\}_{i=k-s}^{k}$, with their respective residuals $\{R_i\}_{i=k-s}^{k}$ belonging to $G_j$, IDR(s) creates $R_{k+1}$ in $G_{j+1}$ and the approximation $X_{k+1}$ using the recursions

$$
X_{k+1} = X_k + \omega_{j+1}V_k + \sum_{i=1}^{s} \gamma_iU_{k-i},
$$

$$
R_{k+1} = V_k - \omega_{j+1}A(V_k), \quad \text{and}
$$

$$
V_k = R_k - \sum_{i=1}^{s} \gamma_iG_{k-i},
$$

where $\{G_i\}_{i=k-s}^{k} \in G_j$ and $U_{k-i} = A(G_{k-i})$. The coefficient $\{\gamma_j\}_{j=1}^{s}$ are obtained by imposing the condition that

$$
V_k \perp P,
$$

which is equivalent to solving the $s \times s$ system of linear equations,

$$
M\mathbf{c} = \mathbf{f},
$$

where $M_{i,j} = \langle P_i, G_{k-s+(j-1)} \rangle_F$ and $\mathbf{f}_i = \langle P_i, R_k \rangle_F$.

Using the fact that $G_{j+1} \subset G_j$, IDR(s) repeats the calculation above to generate $s + 1$ residuals in $G_{j+1}$ with their corresponding approximations. Then, it is possible to create new residuals in the subsequent space $G_{j+2}$. The parameter $\omega_j$ might be chosen freely for the first residual in $G_j$, but the same value should be used for the next residuals in the same space. There exist different options to select the parameter $\omega_j$, see for example [24], [25], and [26].

Equivalent to [1], we can select directly $G_i = -(R_i - R_{i-1})$ and $U_i = X_i - X_{i-1}$. A more general approach to select $G_i$ was presented in [25]. Assuming that $t$ matrices were already created in $G_{j+1}$ with $1 \leq t < s + 1$, then any linear combination of these matrices is also in $G_{j+1}$. In order to create the residual $R_{k+t+1}$ in $G_{j+1}$, they first select vectors $G_i$ as,

$$
G_{k+t} = -(R_{k+t} - R_{k+t-1}) - \sum_{i=1}^{t-1} \beta_iG_{k+i}.
$$
Different choices of these parameter yields different variants of IDR(s) for solving system of linear equations. Normally, the values β’s are chosen to improve the convergence or stability of the IDR(s), for example, in [25] the authors propose the biorthogonal residual variant of IDR(s) selecting β’s such that the $G_{k+t}$ is orthogonal to $P_1, P_2, \ldots, P_{t-1}$. A quasi-minimal residual variant of IDR(s) was proposed in [19], choosing the parameters β to force $G_{k+t}$ to be orthogonal to $G_1, G_2, \ldots, G_{t-1}$. In this work we implement the biorthogonal residual IDR(s), see [25] for more details.

3 Preconditioning

The use of preconditioners in iterative methods is a key element to accelerate or ensure the convergence. However, in the context of solving linear matrix equations $A(X) = C$, there is not a straightforward definition of the application of a preconditioner. An option for applying the preconditioning operation to $V$ is to obtain an approximation to the problem

$$A(X) = V.$$  

For example in the case of the Sylvester equation, the preconditioner applied to $V$ computes an approximate solution of

$$AX + XB = V.$$  

In next section, we present a simple preconditioner for the Sylvester equation based on fixed-point iteration.

3.1 Fixed-point (FP) and Inexact Fixed-point (FP-ILU) preconditioners for the Sylvester equation

In this section we present a simple preconditioning scheme for the iterative method to solve the Sylvester equation,

$$AX + XB = V. \quad (7)$$

The solution of equation (7) is also the solution of the fixed-point iteration,

$$AX_{k+1} = -X_kB + V \quad (8)$$

We propose as preconditioner a few steps of the fixed-point iteration (8). If matrix $A$ is difficult to invert, we propose the application of few steps of the following iteration,

$$M \hat{X}_{k+1} = -\hat{X}_kB + V, \quad (9)$$

where $M$ is an approximation to the matrix $A$. This can be considered as an inexact fixed-point iteration. Particularly, we approximate $A$ using the Incomplete LU factorization. Fixed-point iterations (8) and (9) do not have the same solution. However, if it is assumed that $M$ is a good approximation of $A$ and equation (7) is well-conditioned, one can expect that the solution of the fixed point iteration (9) is close to the solution of the Sylvester equation (7).

We use as preconditioning operator $M(V)$ the fixed-point iteration (9), or if it is possible to solve block linear system with $A$ efficiently, we use iteration (8). The fixed point iteration (8) for solving Sylvester equation has been analyzed in [27]. A sufficient condition for the iteration (8) to converge to its fixed point is that $\|A^{-1}\|\|B\| < 1$ when $A$ is non-singular. Using
this result, it is easy to see that the inexact iteration also converge to its fixed point if $M$ is non-singular and $\|M^{-1}\| B < 1$. For this reason, we can compute $M = LU + E$, the incomplete LU factorization of $A$, using strategies based on monitoring the growth of the norm of the inverse factors of $L$ and $U$ like [28, 29], or scaling matrices such that $\|M^{-1}\| B < 1$ is satisfied.

4 Numerical examples

In all of the following examples, we use as stopping criteria,

$$\frac{\|C - A(X)\|_F}{\|C\|_F} \leq 1e^{-8}.$$ 

**Example 1:** (Solving a Lyapunov equation) In this example, we solve the Lyapunov equation using IDR(s) for $A(X) = C$ with $A(X) = AX + XA^T$. We compare the IDR(s = 4) for matrix equations with Bi-CGSTAB [30] and GMRES [17]. As matrix $A$, we choose the negative of the anti-stable matrix CDDE6 from the Harwell-Boeing collection, and matrix $C = cc^T$, with $c = \text{rand}(961,1)$. Although for IDR(s) the solution of the Lyapunov equation takes more iteration (165 iterations), this is the faster method for CPU-time. IDR(s) consumes 9.33 secs. of CPU time, while GMRES runs in 18.63 secs (131 iterations) and Bi-CGSTAB takes 15.09 secs (566 iterations).

Figure 1: (Example 1) (a) Residual norm for IDR(s = 4), Bi-CGSTAB, and GMRES solving a Lyapunov equation.

**Example 2:** (Solving a time-dependent linear system) We consider the time-dependent linear system,

$$\frac{dy}{dt} = Ay + g(t), \quad t_0 \leq t \leq t_m \quad \text{with} \quad y(t = t_0) = y_0.$$ 

Solving (10) with backward Euler with constant time-step $\delta t$, we obtain a Sylvester equation,

$$-AY + \frac{D}{\delta t} = G + \frac{y_0}{\delta t}e_i^T,$$ 

(11)
where $G = [g(t_1), g(t_2), \ldots, g(t_m)]_{n \times m}$, $D$ is the upper and bidirectional matrix,

$$D = \begin{bmatrix}
1 & -1 & 0 & \ldots & 0 \\
0 & 1 & -1 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \ldots & 1
\end{bmatrix}_{m \times m},$$

and $e_1$ represents the first canonical vector of order $m$. Specifically, We consider the 1D time-dependent convection-diffusion equation,

$$\frac{du}{dt} - \epsilon \frac{d^2 u}{dx^2} + \omega \frac{du}{dx} = 0, \quad 0 \leq t \leq 1, \quad u_0 = 1,$$

(12) with convection parameter $\omega = 1.0$ and diffusion term $\epsilon = 10^{-3}$, $x \in [0, 100]$, with Dirichlet boundary conditions. We discretized this equation using the central finite differences and Euler backward for time integration, with $\delta t = 0.05$ ($m = 20$), $\delta x = 0.1$ ($A \in \mathbb{C}^{1000 \times 1000}$). Figure 2 show the evolution of the residual norm for IDR($s$) and different Krylov method without and with preconditioner (8) respectively.

**Example 3:** (Solving a multi-shift linear system) Solving a multi-shift linear system of equation,

$$(A - \sigma_i I)x = b,$$  \hspace{1cm} for $i = 1, 2, \ldots, m,$  \hspace{1cm} (13)

can also be rewritten as a Sylvester equation,

$$AX - XD = bu^T,$$  \hspace{1cm} (14)

where $D = diag([\sigma_1, \sigma_2, \ldots, \sigma_m])$, $X \in \mathbb{C}^{n \times m}$, and $u = [1, 1, \ldots, 1]^T \in \mathbb{C}^m$. We consider an example presented in [19]. We discretized the convection-diffusion-reaction equation,

$$-\epsilon \Delta u + v^T \nabla u - ru = f$$

with $\epsilon = 1$, $v = [0, 250/\sqrt{5}, 500/\sqrt{5}]^T$, $r = \{0.0, 0.0, 0.0\}$, and homogeneous Dirichlet boundary conditions in the unit cube using central finite differences obtaining a matrix $A$ of size 59319 $\times$ 59319. The right-hand-side vector is defined by the solution $u(x, y, z) = x(1-x)y(1-y)z(1-z)$. Figures 3 shows the behavior of the relative residual norm for GMRES, IDR($s$), and Bi-CGSTAB with and without preconditioner.
Figure 3: (Example 3) (a) Residual norm for IDR(s=2), Bi-CGSTAB, and GMRES solving a Sylvester equation. (b) Residual norm for the preconditioned IDR(s=2), Bi-CGSTAB, and GMRES using two steps of (9).

4.1 More realistic examples

Previous numerical examples are rather academic, in this section we present a pair of more realistic examples.

Example 4: We consider a convection-diffusion problem from ocean circulation simulation. Following model,

\[-r \Delta \psi - \beta \frac{\partial \psi}{\partial x} = (\nabla \times F)_z \quad \text{in } \Omega \tag{15}\]

describes the steady barotropic flow in a homogeneous ocean with constant depth. The function \(\psi\) represents the stream function, \(r\) is the bottom friction coefficient, \(\beta\) is the Coriolis parameter, and \(F\) is given by

\[F = \frac{\tau}{\rho H}, \tag{16}\]

where \(\tau\) represents the external force field caused by the wind stress, \(H\) the average depth of the ocean, and \(\rho\) the water density. The stream function is constant on continent boundaries, i.e.,

\[\psi = C_k \quad \text{on } \partial \Omega_k \quad \text{for } k = 1, 2, \ldots, K, \tag{17}\]

with \(K\) is the number of continents. The values of \(C_k\) are determined by the integral condition,

\[\oint_{\partial \Omega_k} r \frac{\partial \psi}{\partial n} ds = -\oint_{\partial \Omega_k} F \times s ds. \tag{18}\]

We discretize Eqs. (15)–(18) using the finite elements technique described in [31]. The physical parameters used can also be found in the same reference. We obtain a coefficient matrix \(A\) of order 42248, and we have to solve sequence of twelve systems of linear equations,

\[Ax_i = b_i \quad \text{with } i = 1, 2, \ldots, 12. \tag{19}\]

Each of this system of linear equations represent the data for each month of the year. We compare the time for solving (19) using two approaches, solving all the linear systems separately, and solving (19) as a linear matrix equation (a block linear system). In all the cases, we applied incomplete LU with no fill-in as preconditioner. Table 3 shows the time comparison between
Table 1: (Example 4) Comparison of solving (19) as a sequence of linear system or as a matrix equation (a block linear system).

<table>
<thead>
<tr>
<th>Method</th>
<th>Solving (19) separately</th>
<th>Solving (19) as a block linear system</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time [s]</td>
<td># Mat-Vec</td>
</tr>
<tr>
<td>IDR(s = 4)</td>
<td>20.28</td>
<td>2498</td>
</tr>
<tr>
<td>Bi-CGSTAB</td>
<td>25.28</td>
<td>3124</td>
</tr>
<tr>
<td>Bi-CG</td>
<td>29.76</td>
<td>4070</td>
</tr>
<tr>
<td>GMRES(100)</td>
<td>49.17</td>
<td>4200</td>
</tr>
</tbody>
</table>

Figure 4: (Example 4) Solution of the ocean problem.

Example 5: We consider the wedge problem introduced in [32]. This problem is an example of acoustic wave propagation modeled by the Helmholtz equation,

$$ - \Delta p(x) - \left( \frac{2\pi f_k}{c(x)} \right)^2 p(x) = \delta(x - x_s) \quad \text{on } \Omega, $$

with the Sommerfeld boundary conditions,

$$ \frac{\partial p(x)}{\partial n} + i \left( \frac{2\pi f_k}{c(x)} \right) p(x) = 0 \quad \text{on } \partial \Omega. $$

In Eqs. (20) and (21), $p$ represents the acoustic pressure. The computational domain $\Omega$ is $[0, 600] \times [0, 1000]$, and this is divided into three layers, each of them with a different sound velocity given by $c(x)$ (see Figure 5(a)). The source point $x_s$ is located at $(300, 0)$, and the wave frequency $f_k$ is selected as $\{1, 2, 4, 8, 16, 32\}$. We discretized (20)–(21) using finite elements ($120 \times 200$ points), and we obtain the following matrix equation,

$$ KX + iCXD - MXD^2 = bu^T $$

(22)
where $K$ is the stiffness matrix, $C$ represents the boundary conditions, $M$ represents the mass matrix, the $b$ is the discretization of the source term, $D$ is a diagonal matrix, and $u = [1, 1, \ldots, 1]^T_6$. Table 3 shows the comparison between Bi-CG, Bi-CGSTAB, GMRES(200), and IDR($s = 4$) solving the multi-frequency wedge problem.

<table>
<thead>
<tr>
<th>Method</th>
<th>CPU Time [s]</th>
<th>Iterations</th>
<th>Relative residual</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDR($s = 4$)</td>
<td>364.75</td>
<td>38034</td>
<td>7.89e-09</td>
</tr>
<tr>
<td>GMRES(200)</td>
<td>616.11</td>
<td>45366</td>
<td>9.98e-09</td>
</tr>
<tr>
<td>Bi-CGSTAB</td>
<td>582.98</td>
<td>65992</td>
<td>9.34e-09</td>
</tr>
<tr>
<td>Bi-CG</td>
<td>384.60</td>
<td>54338</td>
<td>9.46e-09</td>
</tr>
</tbody>
</table>

Table 2: (Example 5) Comparison of Bi-CG, Bi-CGSTAB, GMRES(200), and IDR($s = 4$) solving the multi-frequency wedge problem.

## 5 Software implementation

The IDR($s$) method for matrix equation is implemented in Python 2.7. The main advantages of using Python are the code portability and ease of use. The software solves general linear matrix equations (1), it only uses the standard Python libraries Numpy v1.8.2 and Scipy v0.14. For compatibility reasons with the Krylov subspace methods implemented in the library Scipy, the software interface is the following:

```python
X, info = idrs(A, C, X0=None, tol=1e-8, s=4, maxit=2000,
               M=None, callback=None).
```

Table 3 describes the input and output parameters.

Following, we illustrate the use of the Python's `idrs` function. The user has to provide an object which define the application of a linear operator $A$ over a matrix $X$, in this case we use example 1 of the numerical tests.
**Parameter** | **Description**
--- | ---
A | (Input object) Object with a *dot* method which defines the linear operator of the problem.
C | (Input matrix) The right hand side matrix.
X0 | (Input matrix) Initial guess.
tol | (Input float) Tolerance of the method.
s | (Input integer) Size of IDR recursion. Bigger *s* gives faster convergence, but also makes the method more expensive.
maxit | (Input integer) Maximum number of iterations to be performed.
M | (Input object) Object with a *solve* method which defines the preconditioning operator of the problem.
callback | (Input function) User-supplied function. It is called as *callback*(X) in each iteration.
X | (Output matrix) Approximate solution.
info | (Output integer) 0 if convergence archived. > 0 if convergence not archive.

Table 3: Parameter of the Python’s *idrs* function

```python
import scipy.io as sio
import numpy as np
from solver import *  # IDR solver package

A = -sio.mmread('cdde6.mtx').tocsr()
n = A.shape[0]
c = np.random.rand(n, 1)
C = c.dot(c.T)
class LyapOp:
    # Defining Linear Operator
def _init_(self, A):
    self.A = A
def dot(self, X):
    return self.A.dot(X) + (self.A.dot(X.T)).T

Aop = LyapOp(A)
X, info = idrs(Aop, C)
```

### 6 Conclusions

In this work we have presented a generalization of the IDR(s) theorem [1] valid for any finite-dimensional space. Using this generalization, we have presented a framework of IDR(s) for solving linear matrix equations. This document also presents several numerical examples of IDR(s) solving linear matrix equations, among them, the most common linear matrix equations like Lyapunov and Sylvester equation. In the examples solving Lyapunov and Sylvester equations, full GMRES required less iteration to converge than IDR and Bi-CGSTAB. However, IDR(s) presented a better performance in CPU time.

Additionally, we present two preconditioners based on fixed point iteration to solve the Sylvester equation. The first preconditioner is fixed-point iteration,

\[ AX_{k+1} = -X_k B + C, \]

that required the explicit inverse or the solving block linear systems with matrix \( A \). Whenever
IDR(s) for solving linear matrix equations

Astudillo and Van Gijzen

is not possible the inversion or solving block linear system with matrix $A$ in an efficient way, we use the inexact iteration

$$MX_{k+1} = -X_kB + C,$$

where $M = LU$ the incomplete LU factorization of $A$. Numerical experiments conducted show a competitive behavior of IDR(s) for solving linear matrix equations.

7 Code availability

Implementations of the Induced Dimension Reduction (IDR(s)) in different programming languages like Matlab, Fortran, Python and Julia are available to download in the web page

http://ta.twi.tudelft.nl/nw/users/gijzen/IDR.html

References


