Induced Dimension Reduction Method for Solving Linear Matrix Equations

Astudillo Rengifo, Reinaldo; van Gijzen, Martin

DOI
10.1016/j.procs.2016.05.313

Publication date
2016

Document Version
Final published version

Published in
Procedia Computer Science

Procedia Computer Science

Volume 80, 2016, Pages 222–232

ICCS 2016. The International Conference on Computational Science

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
R.A.Astudillo@tudelft.nl and M.B.vanGijzen@tudelft.nl

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) [13]) to approximate the solution of linear matrix equations,

$$\sum_{j=1}^{k} A_j X B_j^T = C, \tag{1}$$

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 $vec(X)$ as the vector of order $n \times m$ created by stacking the columns of the matrix X, we can write (1) as,

$$\left(\sum_{j=1}^{k} B_k \otimes A_k\right) vec(X) = vec(C). \tag{2}$$

 Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2016

doi:10.1016/j.procs.2016.05.313

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,

$$AX + XB = C, \tag{3}$$

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,

$$AX + XA^T = C, \tag{4}$$

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 [10] 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 parameterized 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-dependent linear systems, and convection-diffusion equations.

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 [13] as an iterative method to solve large, sparse and non-symmetric system of linear equations

$$A\mathbf{x} = \mathbf{b}, \tag{5}$$

where $A \in \mathbb{C}^{n\times n}$ is the coefficient matrix, $\mathbf{b}$ is the right-hand side vector in $\mathbb{C}^n$, and $\mathbf{x} \in \mathbb{C}^n$ is unknown. IDR(s) has been adapted to solve other related problems like solving block linear systems [5], multi-shift linear systems [15, 2], and eigenvalue problems [6, 1]. 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 $\mathbf{A}_i$. Greek lower case letters represent scalars. $\| \star \|$ represents the classical Euclidean norm, and $\| \star \|_F$ 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 [13]. 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 $\mathcal{D}$ with $\mathcal{A}$ as linear operator defined on the same linear subspace. Corollary 1.1 summarizes this result.

Corollary 1.1. *Let $\mathcal{A}$ be any linear operator over a finite dimensional subspace $\mathcal{D}$ and $\mathcal{I}$ the identity operator over the same subspace. Let $\mathcal{S}$ be any (proper) subspace of $\mathcal{D}$. Define $\mathcal{G}_0 \equiv \mathcal{D}$, if $\mathcal{S}$ and $\mathcal{G}_0$ do not share a nontrivial invariant subspace of the operator $\mathcal{A}$, then the sequence of subspace $\mathcal{G}_j$, defined as*

$$\mathcal{G}_j \equiv (\mathcal{I} - \omega_j \mathcal{A})(\mathcal{G}_{j-1} \cap \mathcal{S}) \quad j = 0, 1, 2 \ldots,$$

with ω_j's nonzero scalars, have the following properties,

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

Proof. The proof is analogous to the one presented in [13]. □

In [5], 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 1.1 has a broader scope; we apply this corollary to solve different types of linear matrix equations.

As in [7], we rewrite problems (1) as

$$\mathcal{A}(X) = C, \tag{6}$$

where $\mathcal{A}(X) = \sum_{j=1}^{k} A_j X B_j$. Using Corollary 1.1, we are able to create residuals $R_k = C - \mathcal{A}(X_k)$ of the problem (6) in the shrinking and nested subspaces $\mathcal{G}_j$ and obtain the approximations X_k. Only changing the definition of the operator $\mathcal{A}$ and the subspace $\mathcal{D}$, we are able to approximate the solution of the linear matrix equation using IDR(s). Assuming that the space $\mathcal{S}$ 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 $\mathcal{G}_j$, IDR(s) creates R_{k+1} in $\mathcal{G}_{j+1}$ and the approximation X_{k+1} using the recursions

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

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

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

where $\{G_i\}_{i=k-s}^{k} \in \mathcal{G}_j$ and $U_{k-i} = \mathcal{A}(G_{k-i})$. 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 $\mathcal{G}_{j+1} \subset \mathcal{G}_j$, IDR($s$) repeats the calculation above to generate $s+1$ residuals in $\mathcal{G}_{j+1}$ with their corresponding approximations. Then, it is possible to create new residuals in the subsequent space $\mathcal{G}_{j+2}$. The parameter ω_j might be chosen freely for the first residual in $\mathcal{G}_j$, but the same value should be used for the next residuals in the same space. There exist different options to select the parameter ω_j, see for example [12], [16], and [11].

Equivalent to [13], 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 [16]. Assuming that t matrices were already created in $\mathcal{G}_{j+1}$ with $1 \leq t < s+1$, then any linear combination of these matrices is also in $\mathcal{G}_{j+1}$. In order to create the residual R_{k+t+1} in $\mathcal{G}_{j+1}$, they first select vectors G_i as,

$$G_{k+t} = -(R_{k+t} - R_{k+t-1}) - \sum_{i=1}^{t-1} \beta_j G_{k+i}.$$

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 [16] 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 [15], 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 [16] 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 $\mathcal{A}(X) = C$, 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

$$\mathcal{A}(X) = V.$$

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,

$$AX + XB = V. \tag{7}$$

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

$$AX_{k+1} = -X_k B + V \tag{8}$$

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,

$$M\hat{X}_{k+1} = -\hat{X}_k B + V, \tag{9}$$

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 $\mathcal{M}(V)$ 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 [8]. 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 (9) 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 [3, 4], or scaling matrices such that $\|M^{-1}\|\|B\| < 1$ is satisfied.

4 Numerical examples

In this section we present four numerical experiments to illustrate the numerical behavior of IDR(s) for solving matrix equations and compare it with others block Krylov subspace solvers. The first three problems are illustrative small examples for different types of matrix equations. The fourth example considers a more realistic application the ocean circulation simulation problem [17].

The numerical experiments presented in this section were implemented in Python 2.7 running on GNU/ Debian on an Intel computer with four cores I5 and 32GB of RAM. We use as stopping criterion,

$$\frac{\|C - \mathcal{A}(X)\|_F}{\|C\|_F} \leq 10^{-8}.$$

4.1 Small examples

Example 1: *(Solving a Lyapunov equation)* In this example, we solve the Lyapunov equation using IDR(s) for $\mathcal{A}(X) = C$ with $\mathcal{A}(X) = AX + XA^T$. We compare IDR($s = 4$) for matrix equations with Bi-CGSTAB [14] and GMRES [9]. As matrix A, we choose the negative of the anti-stable matrix CDDE6 from the Harwell-Boeing collection, and matrix $C = \mathbf{c}\mathbf{c}^T$, with $\mathbf{c} = rand(961, 1)$. Although for IDR($s$) the solution of the Lyapunov equation takes more iteration (165), this is the faster method regarding CPU-time. IDR(s) consumes 7.52 secs. of CPU time, while GMRES runs in 17.53 secs. (131 iterations) and Bi-CGSTAB takes 13.32 secs. (566 iterations).

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

$$\frac{d\mathbf{y}}{dt} = A\mathbf{y} + g(t), \qquad t_0 \leq t \leq t_m \qquad \text{with } \mathbf{y}(t = t_0) = \mathbf{y}_0. \tag{10}$$

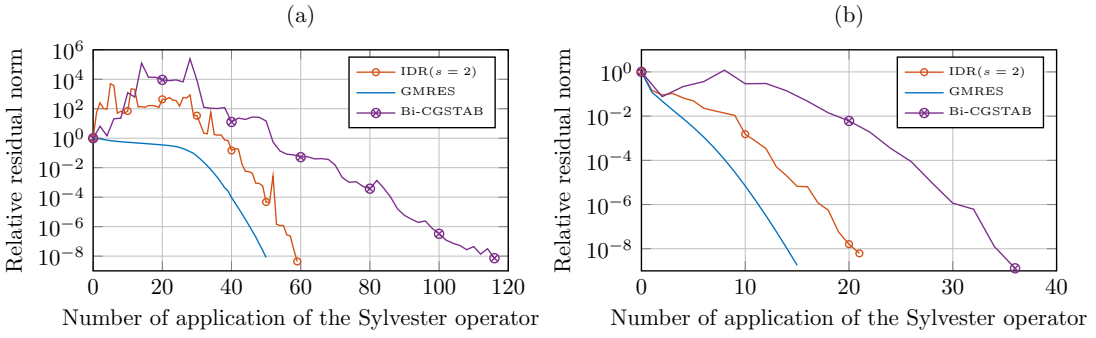

Figure 1: *(Example 2)* (a) Residual norm for IDR(s=2), Bi-CGSTAB, and GMRES solving a Sylvester equation. (b) Residual norm for the preconditioned IDR(s=2), Bi-CGSTAB, and GMRES using two steps of (8).

Solving (10) with backward Euler with constant time-step δ_t, we obtain a Sylvester equation,

$$-AY + Y\frac{D}{\delta_t} = G + \frac{\mathbf{y}_0}{\delta_t}\mathbf{e}_1^T, \tag{11}$$

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

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

and $\mathbf{e}_1$ represents the first canonical vector of order m. Specifically, We consider the 1D time-dependent convection-diffusion equation,

$$\frac{du}{dt} - \epsilon\frac{d^2u}{dx^2} + \omega\frac{du}{dx} = 0, \qquad 0 \le t \le 1, \quad u_{t_0} = 1, \tag{12}$$

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 1 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,

$$(A - \sigma_i I)\mathbf{x} = \mathbf{b}, \qquad \text{for } i = 1, 2, \ldots, m, \tag{13}$$

can also be rewritten as a Sylvester equation,

$$AX - XD = \mathbf{b}\mathbf{u}^T, \tag{14}$$

where $D = diag([\sigma_1, \sigma_2, \ldots, \sigma_m])$, $X \in \mathbb{C}^{n\times m}$, and $\mathbf{u} = [1, 1, \ldots, 1]^T \in \mathbb{C}^m$. We consider an example presented in [15]. We discretize the convection-diffusion-reaction equation,

$$-\epsilon\triangle u + \mathbf{v}^T\nabla u - ru = f$$

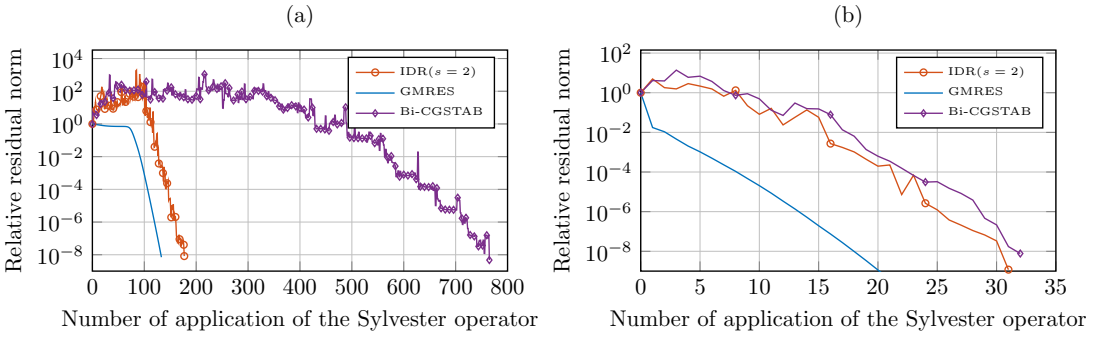

Figure 2: *(Example 3)* (a) Residual norm for IDR(s=2), Bi-CGSTAB, and GMRES solving a Sylvester equation. (b) Residual norm for the preconditioned IDR(s=2), Bi-CGSTAB, and GMRES using two steps of (9).

with $\epsilon = 1$, $\mathbf{v} = [0, 250/\sqrt{5}, 500/\sqrt{5}]^T$, $r = \{0.0, 200.0, 400.0, 600.0, 800.0, 1000.0\}$, and homogeneous Dirichlet boundary conditions in the unit cube using central finite differences obtaining a matrix A of size 59319×59319. The right-hand-side vector is defined by the solution $u(x, y, z) = x(1-x)y(1-y)z(1-z)$. Figures 2 shows the behavior of the relative residual norm for GMRES, IDR(s), and Bi-CGSTAB with and without preconditioner.

4.2 A more realistic example

Example 4: The previous numerical examples are rather academic, in this example we consider a more realistic example. We consider a convection-diffusion problem from ocean circulation simulation. The following model,

$$-r \triangle \psi - \beta \frac{\partial \psi}{\partial x} = (\nabla \times F)_z \qquad \text{in } \Omega \tag{15}$$

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

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

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

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

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 [17]. 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 a sequence of twelve systems of linear equations,

$$A\mathbf{x}_i = \mathbf{b}_i \qquad \text{with } i = 1, 2, \ldots, 12. \tag{19}$$

Method	Solving (19) separately		Solving (19) as a block linear system	
	Time [s]	# Mat-Vec	Time [s]	# Mat-Block
IDR($s = 4$)	17.03	2498	12.29	190
Bi-CGSTAB	22.66	3124	15.35	282
Bi-CG	25.58	4070	20.62	338
GMRES(100)	42.82	4200	38.30	400

Table 1: *(Example 4)* Comparison of solving (19) as a sequence of linear system or as a matrix equation (a block linear system). This exemplifies one of the advantages of using a block-solvers over their sequential counterparts, the time reduction due to the extensive use of block subroutines (BLAS Level 3) over several calls to single vectors routines. Mat-Vec indicates the number of matrix-vector products and Mat-Block indicates the number of matrix-block multiplications.

Each of these 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 drop tolerance 10^{-4} as preconditioner. Table 4 shows the time comparison between the different methods using both approaches. Figure 3 shows the solution computed using IDR($s = 4$) for matrix equations.

Degree	Matrix size	Time [s]	# Mat-Block
4	2594×2594	0.02	5
3	4630×4630	0.11	18
2	10491×10491	0.42	29
1	42249×42249	12.29	190

Table 2: *(Example 4)* Solving the ocean model problem as a matrix equation (a block linear system) using IDR($s = 4$) with ILU preconditioner. Degree is the grid size in the ocean model.

In Table 2, the increment in number of matrix-block products is higher than a fact of two if the grid size is halved. This rather disappointing behaviour seems to be caused by the dropping strategy in the ILU preconditioner, which gives a worse performance for increasingly finer grids. To exclude this effect, we next consider diagonal scaling as preconditioner for IDR(s) in Table 3.

Degree	Matrix size	Time [s]	# Mat-Block
4	2594×2594	0.44	557
3	4630×4630	1.36	773
2	10491×10491	5.09	1204
1	42249×42249	58.03	2883

Table 3: *(Example 4)* Solving the ocean model problem as a matrix equation (a block linear system) using IDR($s = 4$) with diagonal preconditioner. One can see a linear increment with a factor of two in the number of matrix-block products required if the grid size is halved.

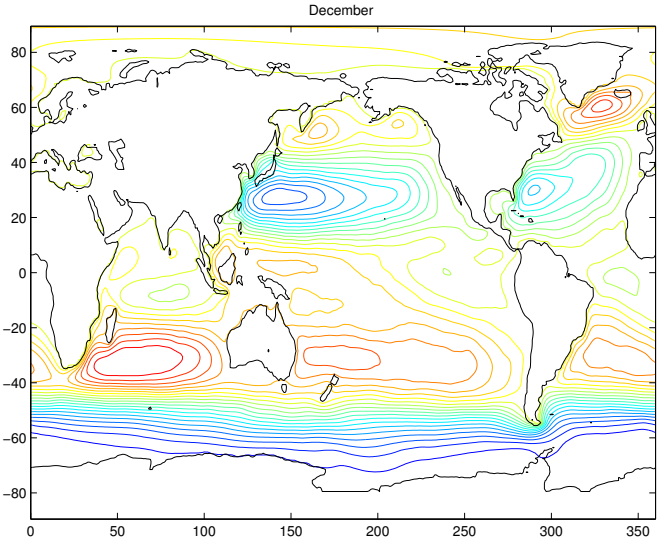

Figure 3: *(Example 4)* Solution of the ocean problem.

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:

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

Table 4 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 $\mathcal{A}$ over a matrix X, in this case we use example 1 of the numerical tests,

```
import scipy.io as io
import numpy as np
from solver import *     # IDRs solver package
A = -io.mmread('cdde6.mtx').tocsr()
n = A.shape[0]
c = np.random.rand(n,1)
C = c.dot(c.T)
class LyapOp:            # Defining Linear Operator
    define __init__(self, A):
        self.A = A
    define dot(self, X):
        return  self.A.dot(X) + (self.A.dot(X.T)).T
Aop = LyapOp(A)
X, info = idrs(Aop,C)
```

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 4: Parameter of the Python's `idrs` function

6 Conclusions

In this work we have presented a generalization of the IDR(s) theorem [13] 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 is not possible the inversion or solving block linear system with matrix A in an efficient way, we use the inexact iteration

$$MX_{k+1} = -X_k B + C,$$

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

[1] R. Astudillo and M. B van Gijzen. A Restarted Induced Dimension Reduction method to approximate eigenpairs of large unsymmetric matrices. *J. Comput. Appl. Math.*, 296:24–35, 2016.

[2] M. Baumann and M. B van Gijzen. Nested Krylov methods for shifted linear systems. *SIAM J. Sci. Comput.*, 37(5):S90–S112, 2015.

[3] M. Bollhöfer. A robust ILU with pivoting based on monitoring the growth of the inverse factors. *Linear Algebra Appl.*, 338(1–3):201–218, 2001.

[4] M. Bollhöfer. A Robust and Efficient ILU that Incorporates the Growth of the Inverse Triangular Factors. *SIAM J. Sci. Comput.*, 25(1):86–103, 2003.

[5] L. Du, T. Sogabe, B. Yu, Y. Yamamoto, and S.-L. Zhang. A block IDR(s) method for nonsymmetric linear systems with multiple right-hand sides. *J. Comput. Appl. Math.*, 235(14):4095–4106, 2011.

[6] M. H. Gutknecht and J.-P. M. Zemke. Eigenvalue Computations Based on IDR. *SIAM J. Matrix Anal. Appl.*, 34(2):283–311, 2013.

[7] M. Hochbruck and G. Starke. Preconditioned Krylov Subspace Methods for Lyapunov Matrix Equations. *SIAM J. Math. Anal. Appl.*, 16(1):156–171, 1995.

[8] M. Monsalve. Block linear method for large scale Sylvester equations. *Comput. Appl. Math.*, 27(1):47–59, 2008.

[9] Y. Saad and M.H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, 1986.

[10] V. Simoncini. Computational methods for linear matrix equations. *To appear in SIAM Rev.*, (To appear).

[11] V. Simoncini and D. B. Szyld. Interpreting IDR as a Petrov-Galerkin method. *SIAM J. Sci. Comput.*, 32(4):1898–1912, 2010.

[12] G. L. G. Sleijpen and H. A. van der Vorst. Maintaining convergence properties of bicgstab methods in finite precision arithmetic. *Numer. Algorithms*, 10(2):203–223, 1995.

[13] P. Sonneveld and M. B. van Gijzen. IDR(s): a family of simple and fast algorithms for solving large nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 31(2):1035–1062, 2008.

[14] H. A. van der Vorst. Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. *SIAM J. Sci. Stat. Comput.*, 13(2):631–644, 1992.

[15] M. B. van Gijzen, G. L. G. Sleijpen, and J.-P. M. Zemke. Flexible and multi-shift induced dimension reduction algorithms for solving large sparse linear systems. *Numer. Linear Algebra Appl.*, 22(1):1–25, 2015.

[16] M. B. van Gijzen and P. Sonneveld. Algorithm 913: An Elegant IDR(s) Variant that Efficiently Exploits Bi-orthogonality Properties. *ACM Trans. Math. Software*, 38(1):5:1–5:19, 2011.

[17] M. B. van Gijzen, C. B. Vreugdenhil, and H. Oksuzoglu. The Finite Element Discretization for Stream-Function Problems on Multiply Connected Domains. *J. Comput. Phys.*, 140(1):30–46, 1998.