Japan Journal of Industrial and Applied Mathematics (2023) 40:95–107
https://doi.org/10.1007/s13160-022-00509-y

ORIGINAL PAPER

# A least squares approach for saddle point problems

**Gul Karaduman**[1] · **Mei Yang**[2] · **Ren-Cang Li**[2]

Received: 14 October 2021 / Revised: 16 March 2022 / Accepted: 28 March 2022 /
Published online: 10 April 2022


**Abstract**

Saddle point linear systems arise in many applications in computational sciences and engineering such as finite element approximations to Stokes problems, image reconstructions, tomography, genetics, statistics, and model order reductions for dynamical systems. In this paper, we present a least-squares approach to solve saddle point linear systems. The basic idea is to construct a projection matrix and transform a given saddle point linear system to a least-squares problem and then solve the least-squares problem by an iterative method such as LSMR: an iterative method for sparse least-squares problems. The proposed method rivals LSMR applied to the original problem in simplicity and ease to use. Numerical experiments demonstrate that the new iterative method is efficient and converges fast

**Keywords** Saddle point problem · Iterative method · Linear system · LSMR · SPPvsLS

## 1 Introduction

In this paper, we consider the saddle point linear system of the form

$$\mathcal{A}z = b, \tag{1a}$$

Supported in part by NSF grants DMS-1719620 and DMS-2009689.

✉ Gul Karaduman
gkaraduman@kmu.edu.tr

Mei Yang
mei.yang@mavs.uta.edu

Ren-Cang Li
rcli@uta.edu

1 Vocational School of Health Services, Karamanoglu Mehmetbey University, Karaman 70200, Turkey

2 Department of Mathematics, University of Texas at Arlington, Arlington, TX 76019-0408, USA

where $\mathcal{A} \in \mathbb{R}^{(n+m)\times(n+m)}$ is a sparse matrix with 2-by-2 block structure, $b \in \mathbb{R}^{n+m}$, and $z \in \mathbb{R}^{n+m}$ is the unknown vector to be found. Specifically, it takes the form

$$\mathcal{A}z \equiv \begin{matrix} & n & m \\ n & A & B_1^{\mathrm{T}} \\ m & B_2 & 0 \end{matrix} \begin{matrix} n \\ m \end{matrix} \begin{bmatrix} x \\ y \end{bmatrix} = b \equiv \begin{matrix} n \\ m \end{matrix} \begin{bmatrix} f \\ 0 \end{bmatrix}, \tag{1b}$$

where $A$ is a sparse matrix. We are interested in the case when $m \ll n$ and for that reason, $B_1$ and $B_2$ may or may not be necessarily sparse. We comment that it is not restrictive to assume that the last $m$ entries of vector $b$ are 0. Later, in Sect. 2, we will remark how to deal with the case when the last $m$ entries of $b$ are not 0.

Iterative solutions for the saddle point problem (1) have been an active research topic in numerical linear algebra. Many applications in computational sciences and engineering applications give rise to saddle point problems such as Stokes problems, flow problems, statistics, image processing, and constrained optimization, see, e.g., [4, 8, 11–16]. In particular, it naturally arises from the following constrained quadratic minimization problem [2, 9]

$$\min \quad h(x) = x^{\mathrm{T}}Ax + x^{\mathrm{T}}B_1^{\mathrm{T}}y - f^{\mathrm{T}}x, \tag{2a}$$

$$\text{subject to} \quad B_2x = 0. \tag{2b}$$

In fact, the corresponding Lagrangian function is

$$\begin{aligned} \mathcal{L}(x,y) &\equiv h(x) + y^{\mathrm{T}}B_2x \\ &= x^{\mathrm{T}}Ax + x^{\mathrm{T}}B_1^{\mathrm{T}}y - f^{\mathrm{T}}x + y^{\mathrm{T}}B_2x \\ &= \begin{bmatrix} x \\ y \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} A & B_1^{\mathrm{T}} \\ B_2 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x \\ y \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} f \\ 0 \end{bmatrix}, \end{aligned}$$

where $y$ is the vector of Lagrangian multipliers. The KKT condition for () is $\nabla\mathcal{L}(x,y) = 0$ which turns out to be the saddle point problem (1) [9]. The reader is referred to the survey article [1] of Benzi, Golub, and Liesen for a more extensive list of the fields where saddle point problems may arise.

The goal of this paper is to seek a least-squares approach. The key idea is to construct a projection matrix and transform the original problem into a least-squares problem and then solve the least-squares problem by an iterative method such as LSMR: an iterative algorithm for sparse least-squares problems [5].

The saddle point problem (1) has a unique solution when $\mathcal{A}$ is nonsingular. A necessary condition for that is $\operatorname{rank}(B_1) = \operatorname{rank}(B_2) = m$. Therefore if either $\operatorname{rank}(B_1) < m$ or $\operatorname{rank}(B_2) < m$, then (1) may or may not have any solution, and when it does have a solution, there are infinitely many solutions. Our proposed method will find a solution to the saddle point problem (1) and it will be in the least-squares sense when no solution exists. Broadly, our method works for the case even if $\mathcal{A}$ is not a square matrix.

Throughout this article, we work with real matrices but the idea can be straightforwardly extended to complex coefficient matrices with minor modifications.

The remainder of this paper is organized as follows. In Sect. 2, we explain how the saddle point problem (1) can be transformed into a least-squares problem. In Sect. 3, we demonstrate the numerical solution of the transformed least-squares problem and present our algorithm. In Sect. 4, we present our numerical results to show the performance of our method for saddle point problems. Finally, some conclusions are made in Sect. 5.

**Notation** $\mathbb{R}^{n\times m}$ is the set of all $n\times m$ real matrices, $\mathbb{R}^n=\mathbb{R}^{n\times 1}$, and $\mathbb{R}=\mathbb{R}^1$. The superscripts "$\cdot^{\mathrm{T}}$" takes the transpose of a matrix or vector. $I_n$ (or simply $I$ if its dimension is clear from the context) is the $n\times n$ identity matrix, and $e_j$ is its $j$th column. For a matrix $X\in\mathbb{R}^{n\times m}$, $\mathcal{R}(X)$ and $\mathcal{N}(X)$ denote the range (column space) and null space of $X$, respectively. Denote by $\|x\|_2$ the Euclidean norm of a vector $x$, and by $\|X\|_1$ the $\ell_1$ operator norm of a matrix $X$.

## 2 Transforming(1)

Let $r=\operatorname{rank}(B_2)\le m$, and perform a QR decomposition with column pivoting [7] on $B_2^{\mathrm{T}}$:

$$B_2^{\mathrm{T}}\Pi=QR, \tag{3}$$

where $Q\in\mathbb{R}^{n\times r}$ has orthonormal columns, i.e., $Q^{\mathrm{T}}Q=I_r$, and $R\in\mathbb{R}^{r\times m}$ is upper triangular (more precisely, trapezoidal), $\Pi\in\mathbb{R}^{m\times m}$ is a permutation matrix (to numerically reveal the rank $r=\operatorname{rank}(B_2)$).

Define

$$P_\perp=I-QQ^{\mathrm{T}}\in\mathbb{R}^{n\times n}, \tag{4}$$

which is the orthogonal projector onto $\mathcal{R}(B_2^{\mathrm{T}})^\perp$, the orthogonal complement of $\mathcal{R}(B_2^{\mathrm{T}})$. When $r=m$, $P_\perp$ can be expressed explicitly in terms of $B_2$ as

$$P_\perp=I-B_2^{\mathrm{T}}(B_2B_2^{\mathrm{T}})^{-1}B_2\in\mathbb{R}^{n\times n}, \tag{5}$$

The next theorem characterizes vectors in $\mathcal{N}(B_2)$.

**Theorem 1** *A vector $x\in\mathcal{N}(B_2)$ if and only if it can be written as $x=P_\perp w$ for some $w\in\mathbb{R}^n$, where $P_\perp$ is as given in (4).*

***Proof*** If $x\in\mathcal{N}(B_2)$, then $B_2x=0$. Thus

$$0=\Pi^{\mathrm{T}}B_2x=R^{\mathrm{T}}Q^{\mathrm{T}}x\quad\Rightarrow\quad Q^{\mathrm{T}}x=0,$$

because $R^{\mathrm{T}}$ has full column rank. Therefore $P_\perp x=(I-QQ^{\mathrm{T}})x=x$, i.e., $x=P_\perp w$ for $w=x$. On the other hand, if $x=P_\perp w$ for some $w\in\mathbb{R}^n$, then

$$\Pi^{\mathrm{T}} B_2 x = \Pi^{\mathrm{T}} B_2 P_{\perp} w = R^{\mathrm{T}} Q^{\mathrm{T}} (I - QQ^{\mathrm{T}}) w$$
$$= R^{\mathrm{T}} (Q^{\mathrm{T}} - Q^{\mathrm{T}} QQ^{\mathrm{T}}) w$$
$$= R^{\mathrm{T}} (Q^{\mathrm{T}} - Q^{\mathrm{T}}) w = 0,$$

as expected. □

Returning to (1), blockwise we have $Ax + B_1^{\mathrm{T}} y = f$ and $B_2 x = 0$. In particular, for any solution $z = \begin{bmatrix} x \\ y \end{bmatrix}$ of (1), we know $x \in \mathcal{N}(B_2)$, or equivalently, $x \in \mathcal{R}(B_2^{\mathrm{T}})^{\perp}$. By Theorem 1, $x$ takes the form $x = P_{\perp} w$ for some $w \in \mathbb{R}^n$. Now substitute $x = P_{\perp} w$ into the saddle point problem (1) to obtain

$$\begin{bmatrix} A & B_1^{\mathrm{T}} \\ B_2 & 0 \end{bmatrix} \begin{bmatrix} P_{\perp} w \\ y \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix}. \tag{6}$$

Since $B_2 P_{\perp} w = 0$ always, the system (6) is equivalent to

$$AP_{\perp} w + B_1^{\mathrm{T}} y = f \quad \Rightarrow \quad \begin{bmatrix} AP_{\perp} & B_1^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} w \\ y \end{bmatrix} = f. \tag{7}$$

This is an under-determined linear system: $n$ equations with $n + m$ unknowns. If it has a solution, it will have infinitely many solutions. In fact, if $\begin{bmatrix} w \\ y \end{bmatrix}$ is a solution, then $\begin{bmatrix} w + q \\ y \end{bmatrix}$ is also one for any $q \in \mathcal{N}(B_2)$. Fortunately, for our purpose, in the end, we set $x = P_{\perp} w$, and so this non-uniqueness does not cause any concern to us at all.

We emphasize that numerically $P_{\perp}$ should never be formed explicitly. If formed, it is usually a dense $n \times n$ matrix, and thus it will take $n^2$ places to store, which can be a burden for large $n$, and a matrix–vector product with $P_{\perp}$ will cost $2n^2$ flops, which is too much of cost. However, if we let $P_{\perp} = I - QQ^{\mathrm{T}}$ exist in this form, then it takes only $rn$ places to store and

$$P_{\perp} w = w - (Q(Q^{\mathrm{T}} w)) \tag{8}$$

can be computed in $(4r + 1)n$ flops which are far less than $2n^2$ flops.

Based on our discussions above, the saddle point problem (1) turned into

$$\min_{w \in \mathbb{R}^n, y \in \mathbb{R}^m} \left\| \begin{bmatrix} AP_{\perp} & B_1^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} w \\ y \end{bmatrix} - f \right\|_2, \tag{9}$$

which is a least squares problem and any solution $\begin{bmatrix} w \\ y \end{bmatrix}$ of it leads to a solution

$$z = \begin{bmatrix} P_{\perp} w \\ y \end{bmatrix}$$

of the original saddle point problem (1). This least-squares problem can be efficiently solved only if matrix–vector products with the coefficient matrix

$$\mathcal{B} := [AP_\perp, B_1^{\mathrm{T}}] \tag{10}$$

can be efficiently implemented. The latter is guaranteed, provided that matrix–vector products with $A$, $B_1$, and $P_\perp$ are efficient to do. Since $A$ and $B_1$ come with the original saddle point problem and thus any numerical method such as LSMR [5] that relies on matrix–vector products with $A$ and $B_1$ is on an equal footing, in this paper, we will not delve into the matrix–vector products with $A$ and $B_1$. Matrix–vector products with $P_\perp$ can be done efficiently according to (8). For future reference, we summarize in Algorithms 1 and 2 how matrix–vector products $\mathcal{B}z$ and $\mathcal{B}^{\mathrm{T}}u$ should be computed.

**Algorithm 1** Efficient Matrix-Vector Product by $\mathcal{B}$ of (10)

**Input:** $z \equiv \begin{bmatrix} w \\ y \end{bmatrix}$, where $w \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$;
**Output:** $\mathcal{B}z$.
1: compute $u = P_\perp w$ as in (8);
2: **return** $z = Au + B_1^{\mathrm{T}} y$.

**Algorithm 2** Efficient Matrix-Vector Product by $\mathcal{B}^{\mathrm{T}}$ of (10)

**Input:** $u \in \mathbb{R}^n$;
**Output:** $\mathcal{B}^{\mathrm{T}} u$.
1: compute $v_1 = A^{\mathrm{T}} u$;
2: compute $v = P_\perp v_1$ as in (8);
3: **return** $\begin{bmatrix} v \\ B_1 u \end{bmatrix}$.

***Remark 1*** Now, we comment on how to deal with the case when the last $m$ entries of $b$ are not 0. Let us say we have $B_2 x = c$, instead of $B_2 x = 0$ as implied by (1). We use (3) to find a particular solution $x_0 = QR^{-\mathrm{T}}\Pi^{\mathrm{T}} c$ to $B_2 x = c$. Perform change of variable $x = x_0 + \tilde{x}$ to get

$$\begin{bmatrix} A & B_1^{\mathrm{T}} \\ B_2 & 0 \end{bmatrix} \begin{bmatrix} \tilde{x} \\ y \end{bmatrix} = \begin{bmatrix} f - Ax_0 \\ 0 \end{bmatrix},$$

which is in the form of (1).

## 3 Solve (9) iteratively

We will solve the under-determined least-squares problem (9) by using a Krylov subspace-based method. In principle, any such method would do. Most recently, LSMR appears to be rather more efficient than any other. For this reason, we will be focusing on using LSMR to solve (9) as an illustration.

Given an initial guess $\begin{bmatrix} w_0 \\ y_0 \end{bmatrix}$, perform change of variable

$$\begin{bmatrix} w \\ y \end{bmatrix} = \begin{bmatrix} w_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} \tilde{w} \\ \tilde{y} \end{bmatrix} \tag{11}$$

to transform (9) to

$$\min_{\tilde{w}\in\mathbb{R}^n,\, \tilde{y}\in\mathbb{R}^m} \left\| \begin{bmatrix} AP_\perp & B_1^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \tilde{w} \\ \tilde{y} \end{bmatrix} - r_0 \right\|_2, \tag{12}$$

where $r_0 = \mathcal{B} \begin{bmatrix} w_0 \\ y_0 \end{bmatrix} - f$. LSMR is an iterative solution technique for sparse least-squares problems. It is based on the Golub-Kahan bidiagonalization [6] which is a recursive procedure to partially bi-diagonalize, in our case, $\mathcal{B} = [AP_\perp, B_1^{\mathrm{T}}]$. Once (12) is solved, the solution to (9) can be recovered by (11).

The Golub-Kahan bidiagonalization [6] on $\mathcal{B} = [AP_\perp, B_1^{\mathrm{T}}]$ goes as follows:

1. set $\beta_1 = \|r_0\|_2, u_1 = r_0/\beta_1, \hat{v}_1 = \mathcal{B}^{\mathrm{T}} u_1, \alpha_1 = \|\hat{v}_1\|_2, v_1 = \hat{v}_1/\alpha_1$;
2. for $i = 1, 2, \dots, k$ do

$$\begin{aligned} \hat{u}_{i+1} &= \mathcal{B}v_i - \alpha_i u_i, \quad \beta_{i+1} = \|\hat{u}_{i+1}\|_2, \quad u_{i+1} = \hat{u}_{i+1}/\beta_{i+1}, \\ \hat{v}_{i+1} &= \mathcal{B}^{\mathrm{T}} u_{i+1} - \beta_{i+1} v_i, \quad \alpha_{i+1} = \|\hat{v}_{i+1}\|_2, \quad v_{i+1} = \hat{v}_{i+1}/\alpha_{i+1}. \end{aligned}$$

Here $k$ is the number of the bidiagonalization step, usually as a parameter that has to be preset. The process runs to its completion if no breakdown occurs, i.e., all $\beta_i > 0$ and $\alpha_i > 0$. We have

$$\mathcal{B}V_k = U_{k+1}F_k, \quad \mathcal{B}^{\mathrm{T}} U_{k+1} = V_k F_k^{\mathrm{T}} + \alpha_{k+1} v_{k+1} e_{k+1}^{\mathrm{T}}, \tag{13}$$

where $V_k = \begin{bmatrix} v_1 & v_2 & \cdots & v_k \end{bmatrix}, U_k = \begin{bmatrix} u_1 & u_2 & \cdots & u_k \end{bmatrix}, U_k^{\mathrm{T}} U_k = I, V_k^{\mathrm{T}} V_k = I$, and

$$F_k = \begin{bmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_k & \alpha_k \\ & & & \beta_{k+1} \end{bmatrix}.$$

Two major actions in performing the Golub-Kahan bidiagonalization are matrix–vector products by $\mathcal{B}$ and $\mathcal{B}^{\mathrm{T}}$. They can be efficiently done as outlined in Algorithms 1 and 2. It can be seen that

$$\mathcal{R}(V_k) = \mathcal{K}_k\big(\mathcal{B}^{\mathrm{T}}\mathcal{B}, \mathcal{B}^{\mathrm{T}}r_0\big) \\ := \mathrm{span}\big(\mathcal{B}^{\mathrm{T}}r_0, \mathcal{B}^{\mathrm{T}}\mathcal{B}(\mathcal{B}^{\mathrm{T}}r_0), \cdots, (\mathcal{B}^{\mathrm{T}}\mathcal{B})^{k-1}(\mathcal{B}^{\mathrm{T}}r_0)\big), \tag{14}$$

where $\mathcal{K}_k\big(\mathcal{B}^{\mathrm{T}}\mathcal{B}, \mathcal{B}^{\mathrm{T}}r_0\big)$ stands for the $k$th Krylov subspace of $\mathcal{B}^{\mathrm{T}}\mathcal{B}$ on $\mathcal{B}^{\mathrm{T}}r_0$. We are seeking the best approximate solution, in the certain sense, to (12) within $\mathcal{K}_k\big(\mathcal{B}^{\mathrm{T}}\mathcal{B}, \mathcal{B}^{\mathrm{T}}r_0\big)$. By (14), such a solution can be expressed as

$$\begin{bmatrix}\tilde{w}\\ \tilde{y}\end{bmatrix} = V_k t \quad \text{for some } t \in \mathbb{R}^k \text{ to be determined.}$$

For LMSR [5], it is required

$$\min_t \big\|\mathcal{B}^{\mathrm{T}}r\big\|_2 \quad \text{with} \quad r = r_0 - \mathcal{B}\begin{bmatrix}\tilde{w}\\ \tilde{y}\end{bmatrix} = r_0 - \mathcal{B}V_k t. \tag{15}$$

We have to solve (15). To this end, we notice

$$\begin{aligned}
\mathcal{B}^{\mathrm{T}}r &= \mathcal{B}^{\mathrm{T}}r_0 - \mathcal{B}^{\mathrm{T}}\mathcal{B}V_k t \quad \text{(by (15))}\\
&= \beta_1\alpha_1 v_1 - \mathcal{B}^{\mathrm{T}}U_{k+1}F_k t \quad \text{(by (13))}\\
&= \beta_1\alpha_1 v_1 - (V_k F_k^{\mathrm{T}} + \alpha_{k+1}v_{k+1}e_{k+1}^{\mathrm{T}})F_k t\\
&= \beta_1\alpha_1 v_1 - V_{k+1}\begin{bmatrix}F_k^{\mathrm{T}}\\ \alpha_{k+1}e_{k+1}^{\mathrm{T}}\end{bmatrix}F_k t\\
&= V_{k+1}\left(\beta_1\alpha_1 e_1 - \begin{bmatrix}F_k^{\mathrm{T}}F_k\\ \alpha_{k+1}\beta_{k+1}e_k^{\mathrm{T}}\end{bmatrix}t\right).
\end{aligned}$$

Since $V_{k+1}^{\mathrm{T}}V_{k+1} = I_{k+1}$, we find

$$\min_t \big\|\mathcal{B}^{\mathrm{T}}r_k\big\|_2 = \min_t \left\|\tilde{\beta}_1 e_1 - \begin{bmatrix}F_k^{\mathrm{T}}F_k\\ \tilde{\beta}_{k+1}e_k^{\mathrm{T}}\end{bmatrix}t\right\|_2, \tag{16}$$

where $\tilde{\beta}_k = \alpha_k\beta_k$ and $\tilde{\beta}_1 = \alpha_1\beta_1$. LSMR [5] uses the double QR decomposition on $F_k^{\mathrm{T}}F_k$ to solve (16). Let $t_k$ be the minimzer. The $k$th LSMR approximation to (12) is then given by

$$\begin{bmatrix}\tilde{w}_k\\ \tilde{y}_k\end{bmatrix} = V_k t_k.$$

What we have just explained is the basic mathematics behind LSMR and it is not for numerical implementation. In fact, Fong and Saunders [5] designed a very elegant and numerically efficient two-term recursive formulas to generate $\begin{bmatrix}\tilde{w}_k\\ \tilde{y}_k\end{bmatrix}$, without having to store all $v_i$. The interested reader is referred to [5] for details.

A reasonable stopping criterion is

$$\left\|\mathcal{B}^{\mathrm{T}} r_k\right\|_2 = \left\|\mathcal{B}^{\mathrm{T}}\left(r_0 - \mathcal{B}\begin{bmatrix}\tilde{w}_k\\ \tilde{y}_k\end{bmatrix}\right)\right\|_2 \leq \mathrm{tol}\,\|r_k\|_2(\|A\|_1 + \|B_1\|_1 + \|B_2\|_1), \quad (17)$$

where tol is a prescribed tolerance. Ideally, the matrix spectral norms of $A$, $B_1$, and $B_2$ should be used in (17), but they are replaced with the $\ell_1$ operator norms for computational convenience. In designing stopping criterion (17), we take into consideration that roughly $\mathcal{B}^{\mathrm{T}} r_k$ is computed to $\mathcal{B}^{\mathrm{T}} r_k + O(\|\mathcal{B}^{\mathrm{T}}\|_2\|r_k\|_2)\epsilon$ and that $\mathcal{B}^{\mathrm{T}} r_k$ should be 0 at a solution, where $\epsilon$ is the unit machone roundoff.In our numerical tests, tol is set to $10^{-12}$ which is usually sufficient for most applications.

**Algorithm 3** Saddle Point Problem via Least Squares (SPPvsLS)

**Input:** $\mathcal{A}$ and $b$ as in (1b), an initial guess $\begin{bmatrix}w_0\\ y_0\end{bmatrix}$ to (7);

**Output:** an approximate solution $\begin{bmatrix}x\\ y\end{bmatrix}$ to the saddle point problem (1).

1: compute the QR decomposition with column pivoting (3) of $B_2^{\mathrm{T}}$;

2: solve least squares problem (12) by LSMR [5] to find an approximate solution $\begin{bmatrix}\tilde{w}\\ \tilde{y}\end{bmatrix}$;

3: recover an approximate solution $\begin{bmatrix}w\\ y\end{bmatrix}$ to (9) by (11);

4: compute $x = P_{\perp} w$ as in (8).

5: **return** $\begin{bmatrix}x\\ y\end{bmatrix}$.

## 4 Numerical results

In this section, we present numerical experiments to demonstrate the performance of our method SPPvsLS (Algorithm 3) for the saddle point problem (1). For that purpose, we apply SPPvsLS to the induced least-squares problem (9) and compare it with LSMR [5] applied to the original problem (1) directly.

In what follows, we will conduct a brief comparison between SPPvsLS (Algorithm 3) and LSMR for the saddle point problems (1).We choose to compare with LSMR because of the fact that LSMR is one of the methods that are comparable, in simplicity and ease of use, to our (Algorithm 3). The other method is LSQR [10] but LSMR currently is the state-of-the-art.

In Table 1, we estimate the numbers of flops for SPPvsLS and LSMR per iteration for solving (1), where (MV) represents the number of flops by one-matrix–vector

**Table 1** Flops per iteration for SPPvsLS and LSMR

| Method | Flops |
|---|---|
| SPPvsLS | 2 (MV)+12 nm |
| LSMR | 2 (MV)+8 nm |

multiplication with $A \in \mathbb{R}^{n\times n}$, which is usually taken to be twice the number of nonzero entries in $A$. We also assume, for simplicity, that $B_1$ and $B_2$ are dense matrices. At the $k$-th step, we can see that the computations for solving the reduced least squares problems are comparable for both methods because the reduced problems have the same size. When $m \ll n$, often one (MV) is much larger than $O(nm)$ and than the cost for solving the reduced least squares problems, and hence SPPvsLS and LSMR cost about the same per iterative step, making the number of iterative steps by either method a reliable measure as to how expensive each of the method is for comparison purpose. But we point out that SPPvsLS involves the rank-revealing QR decomposition (3) as preprocessing at cost of $O(nm^2)$ that is linear in $n$.

All numerical results shown in this section were obtained using MATLAB. The testing matrices are detailed in Table 2, where the column "nonzero" lists the number of nonzero entries of each $\mathcal{A}$. These testing matrices are taken from Suite Sparse Matrix Collection [3]. Some of them have singular coefficient matrices $\mathcal{A}$, but the associated linear systems are consistent, however, and hence solutions exist. Specifically, $\mathcal{A}$ is singular in `GL6_D_6` with rank 156 and in `GL7d11` with rank 59.

We report the relative residual

$$\frac{\|b - \mathcal{A}z\|_2}{\|b\|_2} \tag{18}$$

to gauge the accuracy of an approximate solution $z$. The trivial initial guess, i.e., the zero vector, is used for all problems. Our stopping criterion is either when the number of iterations reaches 6000 or the relative residual (18) is no bigger than $10^{-12}$.

Figures 1, 2, 3, 4 and 5 plot the convergence histories in terms of the relative residual (18) for approximations by Algorithm 3 and by LSMR on the testing matrices in Table 2.

Table 3 lists the numbers of iterations by SPPvsLS and by LSMR to achieve a relative residual (18) less than or equal to $10^{-12}$, except the one marked by "-" which means that the maximum number 6,000 of iterations is reached without achieving the goal of making the relative residual (18) less or equal to $10^{-12}$.

**Table 2** Testing matrices with full rank $B_2$

| Matrix | $n$ | $m$ | Nonzero | Application |
|---|---|---|---|---|
| `lshape2` | 634 | 179 | 6926 | Statistics |
| `dynamicSoaringProblem_1` | 363 | 284 | 5367 | Optimal control |
| `maxwell3` | 1504 | 481 | 18,598 | Electromagnetics |
| `maxwell4` | 6080 | 1985 | 76,902 | Electromagnetics |
| `navier_stokes_N8` | 352 | 127 | 9372 | Statistics |
| `navier_stokes_N16` | 1472 | 511 | 41,692 | Incompressible flow |
| `stokes_N8` | 352 | 127 | 9372 | Computational fluid dynamics |
| `stokes_N16` | 1472 | 511 | 41,692 | Computational fluid dynamics |
| `GL6_D_6` | 469 | 201 | 2839 | Combinatorial optimization |
| `GL7d11` | 1019 | 60 | 2611 | Combinatorial optimization |

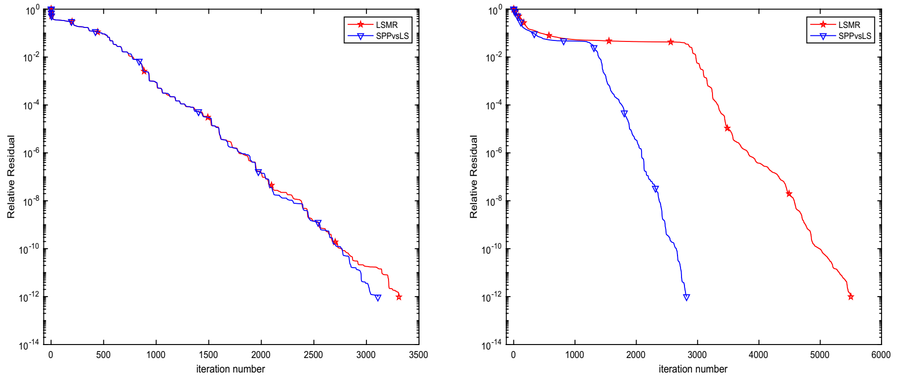


**Fig. 1** Left: Relative residual *vs.* iteration number for `lshape2`; Right: Relative residual *vs.* iteration number for `dynamicSoaringProblem1`

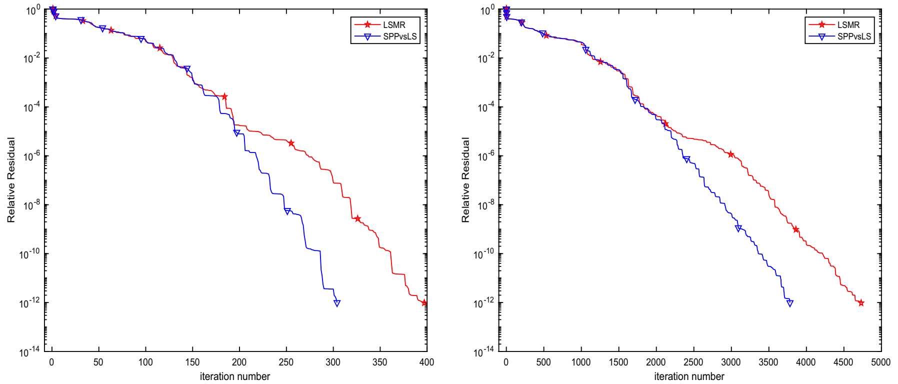


**Fig. 2** Left: Relative residual *vs.* iteration number for `maxwell3`; Right: Relative residual *vs.* iteration number for `maxwell4`

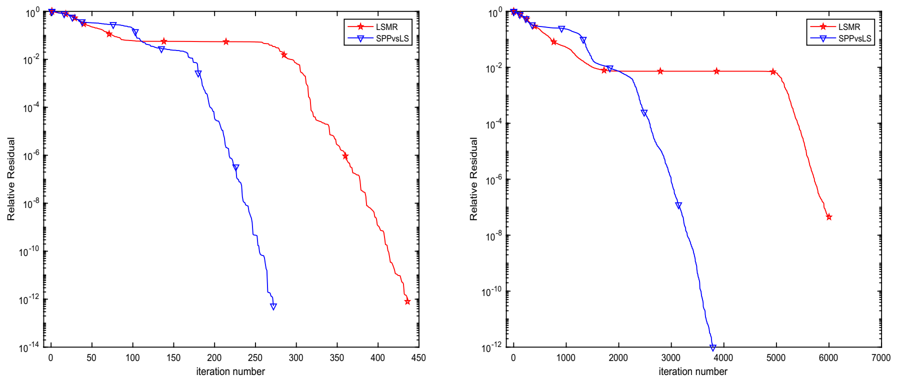


**Fig. 3** Left: Relative residual *vs.* iteration number for `navierstokesN8`; Right: Relative residual *vs.* iteration number for `navierstokesN16`

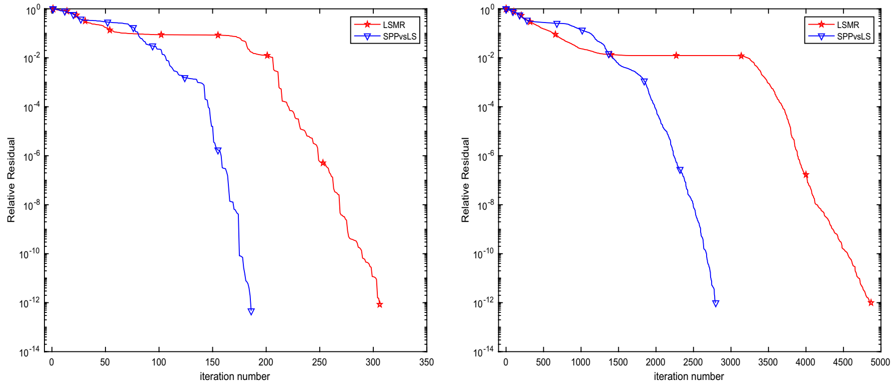


**Fig. 4** Left: Relative residual *vs.* iteration number for `stokesN8`; Right: Relative residual *vs.* iteration number for `stokesN16`

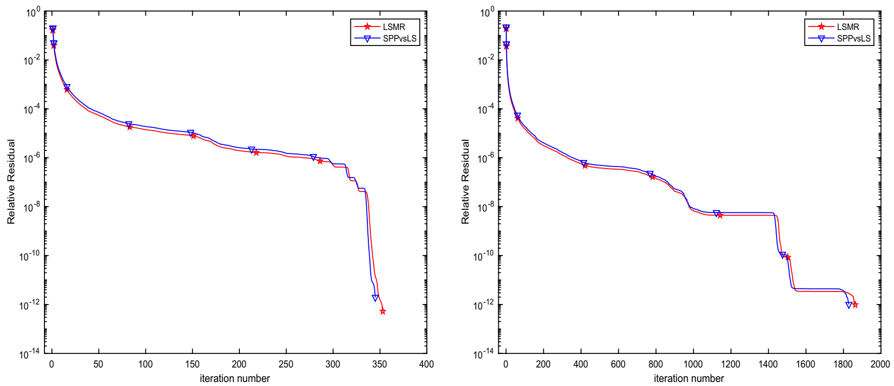


**Fig. 5** Left: Relative residual *vs.* iteration number for `GL6D6`; Right: Relative residual *vs.* iteration number for `GL7d11`

**Table 3** Number of iterations by LSMR and SPPvsLS

| Matrix | LSMR | SPPvsLS |
|---|---|---|
| lshape2 | 3309 | 3109 |
| dynamicSoaringProblem_1 | 5499 | 2820 |
| maxwell3 | 397 | 304 |
| maxwell4 | 4730 | 3781 |
| navier_stokes_N8 | 436 | 272 |
| navier_stokes_N16 | – | 3792 |
| stokes_N8 | 306 | 186 |
| stokes_N16 | 4868 | 2797 |
| GL6_D_6 | 353 | 345 |
| GL7d11 | 1863 | 1829 |

**Table 4** CPU times

| Matrix | LSMR | SPPvsLS |
|---|---|---|
| lshape2 | .16 | .24 |
| dynamicSoaringProblem_1 | .09 | .12 |
| maxwell3 | .28 | .31 |
| maxwell4 | .11 | .46 |
| navier_stokes_N8 | .17 | .10 |
| navier_stokes_N16 | – | .08 |
| stokes_N8 | .07 | .02 |
| stokes_N16 | .94 | .37 |
| GL6_D_6 | .13 | .05 |
| GL7d11 | .19 | .10 |

According to the numerical results obtained from Figs. 1, 2, 3, 4 and 5, we have the following observations:

- In 7 out of 10 examples, SPPvsLS takes much fewer iterations than LSMR to converge. For the other three examples, SPPvsLS performs comparably to LSMR on `GL6_D_6` and `GL7d11`, while still a little better on `lshape2`.
- On `navier_stokes_N16` LSMR fails to make the relative residual (18) less than or equal to $10^{-12}$ within the maximum allowable number of iterations 6000.

In summary, numerical evidence shows that SPPvsLS is in general favored over LSMR.

Next, we compare CPU time by SPPvsLS and LSMR on an Apple laptop with macOS and Intel i7 processor with 2.7 GHz and 8GB memory. Table 4 lists the CPU times by each method in MATLAB to achieve a relative residual less or equal to $10^{-12}$. In 6 out of 10 examples, SPPvsLS wins over LSMR in CPU time. In particular, LSMR fails on `navier_stokes_N16` to produce a sufficiently accurate approximation, within 6,000 iterative steps.

## 5 Conclusions

We have presented an iterative method SPPvsLS for the saddle point linear system in the form of (1) by first constructing a projection matrix to eliminate the last $m$ equations of the system and turn it into a least-squares problem and then solving the resulting least-squares problem by LSMR (or any other iterative method). The method rivals LSMR, a popular iterative method, in their simplicity and ease to use, but more importantly it converges faster, as our numerical examples demonstrate. Because of its simplicity, the method can be easily embedded, by scientists and engineers, into application packages whose intermediate step involves solving large and sparse saddle point linear systems.

**Acknowledgements** The authors would like to thank the anonymous referees for their constructive comments and suggestions that improve the paper.

## References

1. Benzi, M., Golub, G.H., Liesen, J.: Numerical solution of saddle point problems. Acta Numer. **14**, 1–137 (2005)
2. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)
3. Davis, T., Hu, Y.: The University of Florida sparse matrix collection. ACM Trans. Math. Softw. **38**(1), 1–25 (2011)
4. Estrin, R., Greif, C.: On nonsingular saddle-point systems with a maximally rank deficient leading block. SIAM J. Matrix Anal. Appl. **36**(2), 367–384 (2015)
5. Fong, D.C., Saunders, M.A.: LSMR: An iterative algorithm for sparse least-squares problems. SIAM J. Sci. Comput. **33**(5), 2950–2971 (2011)
6. Golub, G.H., Kahan, W.: Calculating the singular values and pseudo-inverse of a matrix. SIAM J. Numer. Anal. **2**, 205–224 (1965)
7. Golub, G.H., Van Loan, C.F.: Matrix Computations, 4th edn. Johns Hopkins University Press, Baltimore (2013)
8. Maryska, J., Rozlozník, M., Tuma, M.: The potential fluid problem and the convergence rate of the minimum residual method. Numer. Linear Algebra Appl. **3**, 525–542 (1996)
9. Nocedal, J., Wright, S.: Numerical Optimization, 2nd edn. Springer, Berlin (2006)
10. Paige, C.C., Saunders, M.A.: LSQR: an algorithm for sparse linear equations and sparse least squares. ACM Trans. Math. Softw. **8**(1), 43–71 (1982)
11. Pearson, J.W., Pestana, J., Silvester, D.J.: Refined saddle-point preconditioners for discretized Stokes problems. Numer. Math. **138**(2), 331–363 (2018)
12. Pestena, J., Rees, T.: Null-space preconditioners for saddle point systems. SIAM J. Matrix Anal. Appl. **37**(3), 1103–1128 (2015)
13. Reid, N.: Saddle point methods and statistical inference. Statist. Sci. **3**(2), 213–227 (1988)
14. Song, Y., Yuen, X., Yue, H.: An inexact Uzawa algorithmic framework for nonlinear saddle point problems with applications to elliptic optimal control problem. SIAM J. Numer. Anal. **57**(6), 2656–2684 (2019)
15. Wu, S.L., Salkuyeh, D.: A shift-splitting preconditioner for asymmetric saddle point problems. Comput. Appl. Math. **39**(4), 314 (2020)
16. Yang, A.L., Li, X., Wu, Y.J.: On semi-convergence of the uzawa-hss method for singular saddle-point problems. Appl. Math. Comput. **252**, 88–98 (2015)