We describe algorithm MINRES-QLP and its FORTRAN 90 implementation for solving symmetric or Hermitian linear systems or least-squares problems. If the system is singular, MINRES-QLP computes the minimum-length solution. In all cases, it circumvents a potential instability in the original MINRES algorithm.

A positive-definite preconditioner may be supplied. Our FORTRAN 90 implementation illustrates a design pattern that allows users to make problem data known to the solver, but hidden and secure from other program units. Moreover, users are spared to program subroutines for reverse communication, which is widely used in scientific computing with FORTRAN 77 but the resulting code usually appears formidable and sacrifices readability.

While we focus on the FORTRAN 90 implementation in this paper, we also provide and maintain FORTRAN 77 and MATLAB 7.8 versions of MINRES and MINRES-QLP.
1. INTRODUCTION

The conjugate-gradient method (CG [?]) and the minimum-residual method (MINRES [?]) are among the most frequently used iterative methods, or Krylov subspace methods, for symmetric nonsingular and compatible linear systems. MINRES-QLP [?, ?] is an extension of MINRES for computing the min-length solution $x$ to the following equations or least-squares (LS) problems:

$$\text{solve } (A - \sigma I)x = b,$$

$$\text{minimize } ||x||_2 \text{ s.t. } (A - \sigma I)x = b,$$

$$\text{minimize } ||x||_2 \text{ s.t. } x \in \arg \min ||(A - \sigma I)x - b||_2,$$

where $A$ is an $n \times n$ symmetric or Hermitian matrix, $I$ is the identity matrix, $\sigma$ is a real or complex shift parameter, and $b$ is a real or complex $n$-vector. The matrix $A$ is usually large and sparse. It is defined by means of a user-written subroutine $A\text{prod}$, whose function is to compute the product $Ay$ for a given vector $y$.

Problems (0???) and (0???) are treated as special cases of (0???). SYMMLQ, MINRES, SQMR [?], and MINRES-QLP are iterative methods specialized for symmetric $A$ and only one product $Ay$ is required in each iteration. SQMR without preconditioning is mathematically equivalent to MINRES. However, for a (singular) least-squares problem SYMMLQ and SQMR could fail while MINRES generally returns only least-squares solutions for (0???); see [?, ?] for examples. To date, MINRES-QLP is probably the most suitable conjugate-gradient type method for solving (0???).

Define $\bar{A} = (A - \sigma I)$. Let $x_k$ be the solution estimate and $r_k = b - \bar{A}x_k$ be the residual vector associated with MINRES-QLP’s $k$th iteration. Without loss of generality, we assume $x_0 = 0$ throughout in this paper. MINRES-QLP provides recurrent estimates of $||x_k||$, $||r_k||$, $||\bar{A}r_k||$, $||\bar{A}||$, $\text{cond}(\bar{A})$, and $||\bar{A}x_k||$, which are used in the iterative method’s stopping conditions. For easy reference, we list the main notation used in this paper in Table 0???.

MINRES-QLP has two phases; it typically starts with MINRES iterations, which we call the MINRES phase, and transfers to the MINRES-QLP phase when a subproblem becomes moderately ill-conditioned. If all the subproblems in MINRES-QLP are well-conditioned, then the problem could be solved entirely in the MINRES phase, in which case the cost of MINRES-QLP is essentially the same as MINRES.

### Table I. Listing of key notation in this paper.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>order of $A$</td>
</tr>
<tr>
<td>$\ell$</td>
<td>last Lanczos iteration when $\beta_{k+1} = 0$</td>
</tr>
<tr>
<td>$\lambda_1, \ldots, \lambda_n$</td>
<td>eigenvalues of $A$</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>scalar shift to diagonal of $A$</td>
</tr>
<tr>
<td>$\bar{A}$</td>
<td>$\bar{A} = A - \sigma I$</td>
</tr>
<tr>
<td>$e_i$</td>
<td>$i$th unit vector</td>
</tr>
<tr>
<td>$| \cdot |$</td>
<td>matrix or vector two-norm</td>
</tr>
<tr>
<td>$\text{cond}(A)$</td>
<td>condition number of $A$ with respect to two norm $= \frac{\max_{i \geq 0}</td>
</tr>
<tr>
<td>$\text{range}(A)$</td>
<td>column space of $A$ defined as ${Ax</td>
</tr>
<tr>
<td>$\text{null}(A)$</td>
<td>null space of $A$ defined as ${x \in \mathbb{R}^n</td>
</tr>
<tr>
<td>$K_k(A, b)$</td>
<td>$k$th Krylov subspace defined as span${b, Ab, \ldots, A^{k-1}b}$</td>
</tr>
</tbody>
</table>
Table II. Comparison of CGLS, LSQR, LSMR, MINRES, and MINRES-QLP. Storage refers to memory required by working vectors in the solvers. Work counts number of floating-point multiplications. MINRES generally returns an arbitrary least-squares solution while the other solvers return the minimum-length solution for problem (0).

<table>
<thead>
<tr>
<th>Method</th>
<th>Storage</th>
<th>Work per iteration</th>
<th>Products per iteration</th>
<th>Linear systems to solve per iteration with preconditioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGLS</td>
<td>4n</td>
<td>5n</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>LSQR</td>
<td>5n</td>
<td>8n</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>LSMR</td>
<td>6n</td>
<td>9n</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MINRES</td>
<td>7n</td>
<td>9n</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MINRES-QLP</td>
<td>7n − 8n</td>
<td>9n − 14n</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

In the MINRES-QLP phase, MINRES-QLP uses 1 more work vector and 5n more multiplications per iteration. We summarize the computational requirements of MINRES, MINRES-QLP, and some of the aforementioned methods in Table ??.

There are problems the more established symmetric methods are preferred:

1. If $\bar{A}$ is known to be positive definite, the conjugate gradient method might be preferable because it requires the same number of iterations as MINRES-QLP but less work per iteration.

2. If $\bar{A}$ is indefinite but $\bar{A}x = b$ is known to have a solution (e.g. if $\bar{A}$ is non-singular), SYMMLQ might be preferred, since it requires the same number of iterations as MINRES-QLP but slightly less work per iteration.

3. If $\bar{A}$ is indefinite and well-conditioned, and $\bar{A}x = b$ has a solution, i.e., it is not a least-squares problem, MINRES might be preferred as it requires the same number of iterations as MINRES-QLP but slightly less work per iteration. SQMR might also be preferred if $\bar{A}$ is indefinite and there is an effective indefinite preconditioner available.

MINRES-QLP described in this paper is implemented in FORTRAN 90 for real double-precision problems. It contains no machine-dependent constants and does not need to use features such as polymorphism from FORTRAN 2003 or 2008. It requires an auxiliary subroutine $\text{Aprod}$ and if a preconditioner is supplied, a second subroutine $\text{Msolve}$. Since FORTRAN 90 contains the intrinsic COMPLEX data type, our implementation is also adapted for complex problems. Precision other than double can be handily obtained by supplying different values to the data attribute KIND. The program can be compiled with FORTRAN 90 and FORTRAN 95 compilers such as $\text{f90}$, $\text{f95}$, $\text{g95}$, and $\text{gfortran}$. We also have FORTRAN 77 and MATLAB implementations, the latter being capable of solving both real and complex problems readily. All versions are available for download at [?].

1.1 Other Iterative Methods

Earlier successful methods for rectangular problems (with $\sigma = 0$) are CGLS and LSQR [?; ?], and LSMR [?], which enjoys the desirable properties of monotonically decreasing residual norms $||r||$ and $||\bar{A}r||$. In fact, CGLS, LSQR, and LSMR are applicable for problems in which $A$ is any general matrix with $m$ rows and $n$ columns (with $\sigma = 0$). However, all three methods require two products $Ay$ and $A^Tz$ in each
iteration. GMRES [?] is designed for square unsymmetric problems. It requires only products $Ay$, but (even with restart) it typically takes significantly more memory for storing many long working vectors and more work for computations of inner products.

We would not discourage using CGLS, LSQR, or LSMR if storage needs to be minimized or the number floating-point multiplications due to an extra product per iteration is less than $12n$. In fact, we would recommend LSQR and LSMR if the goal is to regularize an ill-posed problem using a small damping factor $\lambda \neq 0$ in the following manner:

\[
\minimize ||x|| \quad \text{s.t.} \quad x \in \arg \min \left\| \begin{bmatrix} A \\ \lambda I \end{bmatrix} x - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|.
\]  

(4)

Nonetheless, regularization changes the original problem and the above formulation also destroys the original problem’s symmetry. The normal equation of (0??) is $(A^2 + \lambda^2 I)x = Ab$, which suggests that a diagonal shift to $A$ may well serve the same purpose in some cases. For symmetric positive-definite $A$, $A + \sigma I$ (with $\sigma > 0$) enjoys a smaller condition number. When $A$ is indefinite, there may not exist a good choice of $\sigma$, e.g., if the eigenvalues of $A$ were symmetrically positioned around zero. However, when this symmetric form is applicable, it is convenient in MINRES and MINRES-QLP by design; see (0??), (0??), and (0??). We also remark that MINRES and MINRES-QLP produce good estimates of the largest and smallest singular values of $A$ (via diagonal values of $R_k$ or $L_k$ in (0??) and (0??); see [?, section 4]) Lastly, there are three main regularization tools in existing literature (see [?, sections 12.1.1-12.1.3] and [?): LSQI, cross-validation, and L-curve. LSQI involves solving a nonlinear equation and is not immediately compatible with the Lanczos framework. Cross-validation takes one row out at a time and thus does not preserve symmetry in our problem. L-curve for a CG-type method takes iteration $k$ ($\sigma = 0$) as the regularization parameter [?, Chapter 8] if both $\|r_k\|$ and $\|x_k\|$ are monotonic. Unfortunately, while $\|r_k\|$ is monotonic in MINRES and MINRES-QLP, $\|x_k\|$ is not. We prefer the condition L-curve approach in [?], which graphs $\text{cond}(T_k)$ against $\|r_k\|$. Yet another L-curve feasible in MINRES-QLP is $\|x_{k-2}^{(2)}\|$ against $\|r_k\|$, since the former is also monotonic (but available two iterations in lag); see section 0??.

Both MINRES and MINRES-QLP use the stopping condition $\text{cond}(\hat{A}) \geq A\text{condlim}$, where $A\text{condlim}$ is a user-supplied parameter for limiting the subproblems (0??) and (0??) from growing too ill-conditioned. MINRES-QLP has an additional stopping condition $\|x_k\| \geq \text{maxxnorm}$, where maxxnorm is also a user-supplied parameter. For example, it may be known a priori that $\|x_k\| = 1$ in some Markov chain problems and thus users could specify maxxnorm = 1 in MINRES-QLP. Moreover, MINRES-QLP monitors more carefully than MINRES the rank of the tridiagonal matrix $T_k$ in (0??) by checking $|\gamma_k^{(4)}|$ in (0??), we may say that MINRES-QLP is more rank-revealing and regularization is an even stronger feature in the solver; see [?, Chapter 4] or [?, Section 8] for numerical examples.
2. MATHEMATICAL BACKGROUND

Notation and details of algorithmic development are given in [?, ?] and summarized here.

2.1 The Lanczos Process

MINRES and MINRES-QLP use the symmetric Lanczos process [?] to reduce $\tilde{A}$ to a tridiagonal form $T_k$. The quantities produced from $A$ and $b$ after $k$ steps of the tridiagonalization are

\[ v_0 \equiv 0, \quad \beta_1 = \|b\|, \quad \beta_1 v_1 = b, \]
\[ p_k = Av_k - \sigma v_k, \quad \alpha_k = v_k^T p_k, \quad \beta_{k+1} v_{k+1} = p_k - \alpha_k v_k, \quad \beta_k v_{k-1} = \beta_{k+1} v_k - \beta_k v_{k-1}, \]

where we choose $\beta_k > 0$ to give $\|v_k\| = 1$. Numerically,

\[ p_k = Av_k - \sigma v_k, \quad \alpha_k = v_k^T p_k, \quad \beta_{k+1} v_{k+1} = p_k - \alpha_k v_k \]

is slightly better than (0??) [?]. We could express (0??) in matrix form:

\[ V_k \equiv [v_1 \cdots v_k], \quad AV_k = V_{k+1} T_k, \quad T_k = \begin{bmatrix} T_k & 0 \\ \beta_{k+1} e_1^T \end{bmatrix}. \]

In exact arithmetic, the columns of $V_k$ are orthonormal and the process stops with $k = \ell$ when $\beta_{\ell+1} = 0$ for some $\ell \leq n$, and then $AV_k = V_\ell T_\ell$. The rank of $T_\ell$ could be $k \ell$ or $\ell - 1$ (see Theorem 0?? for details).

2.2 MINRES Phase

MINRES-QLP typically, but not necessarily, starts with a MINRES phase, which applies a series of reflectors $Q_k$ to transform $T_k$ to an upper triangular matrix $R_k$:

\[ Q_k [T_k \quad \beta_1 e_1] = \begin{bmatrix} R_k & t_k \\ 0 & \phi_k \end{bmatrix} \equiv \begin{bmatrix} R_k & \bar{t}_{k+1} \end{bmatrix}, \]

where

\[ Q_k = Q_{k,k+1} \cdots Q_{1,2}, \quad Q_{k,k+1} \equiv \begin{bmatrix} I_{k-1} & c_{k+1} s_k \\ s_{k-1} & -c_{k+1} s_k \end{bmatrix}. \]

In the $k$th step, $Q_{k,k+1}$ is effectively a Householder reflector of dimension 2 [?, Exercise 10.4] and its action including its effect on later columns of $T_\ell$, $k < j \leq \ell$, is compactly described by

\[
\begin{bmatrix}
  c_k & s_k \\
  s_k & -c_k
\end{bmatrix} \begin{bmatrix}
  \gamma_k & \delta_{k+1} & 0 & \cdots \\
  \beta_{k+1} & \alpha_{k+1} & \beta_{k+2} & \cdots \\
  \phi_{k-1} & 0 & \cdots & \cdots & \phi_k
\end{bmatrix} =
\begin{bmatrix}
  \gamma_k^{(2)} & \delta_{k+1}^{(2)} & \epsilon_{k+2} & \cdots \\
  0 & \gamma_{k+1} & \delta_{k+2} & \cdots & \phi_k
\end{bmatrix},
\]

where the superscripts with numbers in parentheses indicate the number of times the values have been modified.

The $k$th solution approximation to (0??) is then defined to be $x_k = V_k y_k$, where $y_k$ solves the subproblem

\[ y_k = \arg \min_{y \in \mathbb{R}^k} \|R_k y - \bar{t}_{k+1}\| = \arg \min_{y \in \mathbb{R}^k} \|T_k y - \beta_1 e_1\|. \]

When $\beta_{k+1} > 0$, $R_k$ is nonsingular and the unique solution of the above subproblem satisfies $R_k y_k = \bar{t}_{k+1}$. Instead of solving for $y_k$, MINRES solves $R_k^T D_k T_k^T = V_k^T$ by forward

substitution, obtaining the last column $d_k$ of $D_k$ at iteration $k$. At the same time, it updates $x_k \in K_k(A, b)$ (see Table 0?? for definition) via $x_0 \equiv 0$ and

$$x_k = V_k y_k = D_k R_k y_k = D_k t_k = x_{k-1} + \tau_k d_k, \quad \tau_k \equiv \epsilon_k^T t_k,$$

where it can be shown that $d_k = \left( v_k - \delta_k^{(2)} d_{k-1} - \epsilon_k d_{k-2} \right) / \gamma_k^{(2)}$.

2.3 MINRES-QLP Phase

In the implementation of MINRES-QLP, the MINRES phase, if it exists, transfers to the MINRES-QLP phase when an estimate of the condition number of $A$ exceeds an input parameter $\text{transcond}$. Thus, $\text{transcond} > 1/\varepsilon$ leads to MINRES iterates throughout (where $\varepsilon \approx 10^{-16}$ denotes the floating-point precision), while $\text{transcond} = 1$ generates MINRES-QLP iterates from the start.

Suppose for now that there is no MINRES phase. Then MINRES-QLP applies left reflections as in (0??) and a further series of right reflections $P_k$ to transform $R_k$ to a lower triangular matrix $L_k = R_k F_k$, where

$$P_k = P_{k,1,2} P_{k,1,3} P_{k,2,3} \cdots P_{k,2,k-1,k},$$

$$P_{k-2,k} = \begin{bmatrix} \epsilon_k & c_k \gamma_k & s_k \end{bmatrix}, \quad P_{k-1,k} = \begin{bmatrix} \epsilon_k & c_k & s_k \\ \gamma_k & s_k & -c_k \\ c_k & s_k & -c_k \end{bmatrix}.$$

In the $k$th step, the actions of $P_{k-2,k}$ and $P_{k-1,k}$ are compactly described by

$$\begin{bmatrix} \gamma_k^{(2)} & \gamma_k^{(4)} \end{bmatrix} = \begin{bmatrix} \epsilon_k & c_k \gamma_k & s_k \end{bmatrix} \begin{bmatrix} 1 & c_k \\ s_k & -c_k \end{bmatrix} \begin{bmatrix} \epsilon_k \gamma_k \end{bmatrix}.$$

The $k$th solution approximation to (0??) is then defined to be $x_k = V_k y_k = V_k P_k u_k = W_k u_k$, where $u_k$ solves the subproblem

$$u_k \equiv \arg \min_u \| u \| \quad \text{ s.t. } u \in \arg \min_{u \in \mathbb{R}^k} \left\| \begin{bmatrix} L_k \\ 0 \end{bmatrix} u - \begin{bmatrix} t_k \end{bmatrix} \right\|_2.$$ 

For $k < \ell$, $R_k$ and $L_k$ are nonsingular because $T_k$ has full column rank by Lemma 0?? in the next subsection. It is only when $k = \ell$ and $b \not\in \text{range}(A)$ that $R_k$ and $L_k$ are singular with rank $\ell - 1$ by Theorem 0??, in which case it can be shown that $\eta_k = \vartheta_k = \gamma_k^{(4)} = 0$ in (0??) and $L_{\ell-1} = \begin{bmatrix} L_{\ell-1} \\ 0 \end{bmatrix}$ with $L_{\ell-1}$ nonsingular. In any case, we only need to solve nonsingular lower triangular systems $L_k u_k = t_k$ or $L_{\ell-1} u_{\ell-1} = t_{\ell-1}$ by forward substitution. Hence $u_k$ and $y_k = P_k u_k$ are indeed the min-length LS solutions of (0??) and (0??) respectively. MINRES-QLP updates $x_{k-2}$ to obtain $x_k$ by short-recurrence orthogonal steps:

$$x_{k-2} = x_{k-3} + \mu_{k-2} u_{k-2}, \text{ where } x_{k-3} = W_{k-3} u_{k-3},$$

$$x_{k-2} = x_{k-2} + \mu_{k-1} u_{k-1}, \text{ where } x_{k-2} = W_{k-2} u_{k-2}.$$

Here \( w_j \) refers to the \( j \)th column of \( W_k = V_k P_k \) and \( \mu_i \) is the \( i \)th element of \( u_k \).

If this phase is preceded by a MINRES phrase of \( k \) iterations for some \( k \) such that \( 0 < k < \ell \), then it starts with transferring the last three basis vectors \( d_{k-2}, d_{k-1}, d_k \) and the solution estimate \( x_k \) from \((0)\) to \( w_{k-2}, w_{k-1}, w_k \), and \( x^{(2)}_{k-2} \) in \((0)\) respectively and after solving the last two equations in \((0)\) for \( \mu_{k-1} \) and \( \mu_k \) using the following relations:

\[
W_k = D_k L_k, \quad x^{(2)}_{k-2} = x_k - w_{k-1} \mu_{k-1} - w_k \mu_k.
\]

Obviously the cheaply available right reflections \( L_k = R_k P_k \) and the last 3 \( \times \) 3 principal submatrix of \( L_k \) need to have been computed in the MINRES phase and stored in memory so that the transfer process can be readily made.

### 2.4 Norm Estimates and Stopping Conditions

The algorithm provides the following short-recurrence norm estimates and their initial values:

\[
\| r_k \| \approx \phi_k = \phi_{k-1} s_{k}, \quad \phi_0 = \| b \|, \quad (\phi_k \searrow) \\
\| \bar{A} r_k \| \approx \psi_k = \phi_k \| [\gamma_{k+1} \delta_{k+2}] \| , \\
\| x_k \| \approx \chi_k = \| [\chi^{(2)}_{k-2} \mu^{(2)}_{k-1} \mu_k] \| , \\
\chi^{(2)}_{k-2} = \| [\chi^{(2)}_{k-3} \mu^{(2)}_{k-2}] \| , \quad \chi_0 = 0, \quad (\chi^{(2)}_k \nearrow) \\
\| \bar{A} x_k \| \approx \omega_k = \| [\omega_{k-1} \tau_k] \| , \quad \omega_0 = 0, \\
\| A \| \geq A_k \approx \max_{i=1, \ldots, k} \{ A_{k-1}, \| T_k e_k \|, L_k(i, i) \} , \quad A_0 = 0, \quad (A_k \nearrow) \\
\text{cond}(\bar{A}) \geq \text{cond}(T_k) \approx \kappa_k = \frac{A_k}{\min_{i=1, \ldots, k} L_k(i, i)} , \quad \kappa_0 = 1. \quad (\kappa_k \nearrow)
\]

The up and down arrows in parentheses indicates that the quantities are monotonic increasing or decreasing if such properties exist. MINRES-QLP has a total of 15 possible stopping conditions in five classes that use the above norm estimates and user-input parameters \( \text{itnlim}, \text{rtol}, \text{Acondlim}, \text{and maxnorm} \):

1. From Lanczos and MINRES-QLP:
   \[
   k = \text{itnlim}; \quad \beta_{k+1} < \varepsilon; \quad \left| \gamma^{(4)}_k \right| < \varepsilon; 
   \]

2. Normwise relative backward errors (NRBE) [7]:
   \[
   \| r_k \| / (\| \bar{A} \| \| x_k \| + \| b \|) \leq \max(\text{rtol}, \varepsilon); \quad \| \bar{A} r_k \| / (\| \bar{A} \| \| r_k \|) \leq \max(\text{rtol}, \varepsilon); 
   \]

3. Regularization attempts:
   \[
   \text{cond}(\bar{A}) \geq \min(\text{Acondlim}, 0.1/\varepsilon); \quad \| x_k \| \geq \text{maxnorm};
   \]
Degenerate cases:
\[ \beta_1 = 0 \quad \Rightarrow \quad b = 0 \quad \Rightarrow \quad x = 0 \text{ is the solution}; \]
\[ \beta_2 = 0 \quad \Rightarrow \quad v_2 = 0 \quad \Rightarrow \quad \bar{A}b = \alpha_1 b, \]
i.e., \( b \) and \( \alpha_1 \) are an eigenpair of \( \bar{A} \), and \( x = b/\alpha_1 \) solves \( \bar{A}x = b \);

(5) Erroneous inputs:
\[ A \text{ not symmetric}; \quad M \text{ not symmetric positive definite}, \]
where \( M \) is a preconditioner to be described in the next section. For symmetry, it is not practical to check \( c_i^T A e_j = c_j^T A e_i \) for all \( i, j = 1, \ldots, n \). Instead, we statistically test if \( z = |x^T(Ay) - y^T(Ax)| \) is sufficiently small, where the value of each component in the \( n \)-vectors \( x \) and \( y \) is drawn from the standard normal distribution. As for positive definiteness of \( M \), since \( M \) is positive definite if and only if \( M^{-1} \) is positive definite, and the latter implies that \( \beta_k = \sqrt{\gamma_k (\bar{A})^{-1}} > 0 \) (see section 0??), we simply perform a necessary test if \( \beta_k > 0 \) in each iteration.

We find that the recurrent relations for \( \phi_k \) and \( \psi_k \) hold to machine accuracy, so \( x_k \) is an acceptable solution of (0??) if the computed NRBE value of either \( \phi_k \) or \( \psi_k \) is suitably small. For other stopping conditions, the final \( x_k \) may not be an acceptable solution.

It is well-known that for most real-world problems with finite precision, the Lanczos stopping condition \( \beta_{k+1} \leq n \| \bar{A} \| \varepsilon \) is rarely observed. In addition, the stopping condition \( \gamma_k (\bar{A}) \) is to handle the case when \( \gamma_k (\bar{A}) \) is numerically small but not exactly zero. For an inconsistent problem, this Lanczos breakdown is amiable and expected by Theorem 0?? in the next section. However, if this condition happens before the NRBE condition \( \| \bar{A}r_k \| / (\| M \| \| r_k \| ) \leq \max(\text{rtol}, \varepsilon) \) is met, it is an indication that the problem is very ill-conditioned, in which case techniques involving regularization and preconditioning in section 0?? may be helpful.

2.5 Two Theorems

We complete this section by presenting two theorems from [?] but here we provide simpler and more direct proofs.

For all \( k < \ell \), \( \beta_1, \ldots, \beta_{k+1} > 0 \) and hence \( T_k \) has full column rank. We have proved the following lemma.

**Lemma 2.1.** \( \text{rank}(T_k) = k \) for all \( k < \ell \).

**Theorem 2.2.** \( T_\ell \) is nonsingular if and only if \( b \in \text{range}(\bar{A}) \). Furthermore, \( \text{rank}(T_\ell) = \ell - 1 \) if \( b \notin \text{range}(\bar{A}) \).

**Proof.** We use \( \bar{A}V_\ell = V_\ell T_\ell \) twice. First, if \( T_\ell \) is nonsingular, we can solve \( T_\ell y_\ell = \beta_1 e_1 \) and then \( \bar{A}V_\ell y_\ell = V_\ell T_\ell y_\ell = V_\ell \beta_1 e_1 = b \). Conversely, if \( b \in \text{range}(\bar{A}) \) then \( \text{range}(V_\ell) \subseteq \text{range}(\bar{A}) \). Note \( T_\ell \) is singular. Then there exists \( z \neq 0 \) such that \( V_\ell T_\ell z = \bar{A}V_\ell z = 0 \). That is, \( 0 \neq V_\ell z \in \text{null}(\bar{A}) \). But this is impossible because \( V_\ell z \in \text{range}(\bar{A}) \) and \( \text{null}(\bar{A}) \cap \text{range}(V_\ell) = 0 \). Thus, \( T_\ell \) must be nonsingular.

We have shown if \( b \notin \text{range}(\bar{A}) \), \( T_\ell = \begin{bmatrix} T_{\ell-1} & \beta_{\ell-1} e_1 \\ \alpha_{\ell-1} e_1 & \alpha_{\ell-1} \end{bmatrix} \) is singular, and therefore \( \ell > \text{rank}(T_\ell) \geq \text{rank}(T_{\ell-1}) = \ell - 1 \) by Lemma 0?? . Therefore, \( \text{rank}(T_\ell) = \ell - 1 \).
By Lemma 0?? and Theorem 0?? we are assured that the QLP decomposition without column pivoting [?, ?] for $T_k$ is rank-revealing in the case of a least-squares problem, and hence the next theorem follows.

**Theorem 2.3.** In MINRES-QLP, $x_t$ is the minimum-length solution of (0??).

**Proof.** $y_t$ comes from the min-length LS solution of $T_t y_t \approx \beta_t e_1$ and thus satisfies the normal equation $T_t^2 y_t = T_t \beta_t e_1$ and $y_t \in \text{range}(T_t)$. Now $x_t = V_t y_t$ and $A x_t = AV_t y_t = V_t A y_t$. Hence $A^2 x_t = AV_t T_t y_t = V_t T_t \beta_t e_1 = \bar{A} b$. Thus $x_t$ is a LS solution of (0??). Since $y_t \in \text{range}(T_t)$, $y_t = T_t z$ for some $z$, and so $x_t = V_t y_t = V_t T_t z = AV_t z \in \text{range}(A)$ is the min-length LS solution of (0??). □

### 3. Preconditioning

Iterative methods can be accelerated if preconditioners are available and well-chosen. For MINRES-QLP, we want to choose a symmetric positive-definite matrix $M$ to solve a nonsingular system (0??) by implicitly solving an equivalent symmetric compatible system $M^{-\frac{1}{2}} A M^{-\frac{1}{2}} \bar{x} = \bar{b}$, where $M^{-\frac{1}{2}} \bar{x} = \bar{x}$, $\bar{b} = M^{-\frac{1}{2}} b$, and $\text{cond}(M^{-\frac{1}{2}} A M^{-\frac{1}{2}}) \ll \text{cond}(A)$. This two-sided preconditioning preserves symmetry. Thus we can derive preconditioned MINRES-QLP by applying MINRES-QLP to the equivalent problem and obtain $x = M^{-\frac{1}{2}} \bar{x}$.

With preconditioned MINRES-QLP, we can solve a singular compatible system (0??), but we will obtain a least-squares solution that is not necessarily the minimum-length solution (unless $M = I$). For incompatible systems (0??), preconditioning alters the “least-squares” norm and the solution is of minimum-length in the new norm space. We refer readers to [?, Section 7] for examples and a detailed discussion of various applicable approaches to preserve the correct “minimum length”.

In preconditioned MINRES-QLP, we define

$$z_k = \beta_k M^{-\frac{1}{2}} v_k, \quad q_k = \beta_k M^{-\frac{1}{2}} v_k,$$

so that $M q_k = z_k$. (13)

Then $\beta_k = \|\beta_k v_k\| = \|M^{-\frac{1}{2}} z_k\| = \|z_k\|_{M^{-1}} = \|q_k\|_{M} = \sqrt{q_k^T z_k}$, where the square root is well defined because $M$ is positive definite, and the following expressions replace the quantities in (0??) in the Lanczos iterations:

$$p_k = A q_k - \sigma q_k, \quad \alpha_k = \frac{1}{\beta_k} q_k^T p_k, \quad z_{k+1} = \frac{1}{\beta_k} p_k - \frac{\alpha_k}{\beta_k} z_k - \frac{\beta_k}{\beta_{k-1}} z_{k-1}. \quad (14)$$

In addition, we need to solve a linear system $M q_k = z_k$ from (0??) in each iteration.

In the MINRES phase, we can define $\bar{d}_k = M^{-\frac{1}{2}} d_k$ and update the solution of the original problem (0??) by

$$\bar{d}_k = \left( \frac{1}{\beta_k} q_k - \beta_k^{-1} d_k - \epsilon_k \bar{d}_{k-2} \right) / \gamma_k^{(2)}, \quad x_k = M^{-\frac{1}{2}} \bar{x}_k = x_{k-1} + \tau_k \bar{d}_k.$$

In the MINRES-QLP phase, we define $W_k \equiv M^{-\frac{1}{2}} W_k = (M^{-\frac{1}{2}} V_k) P_k$ and update the solution estimate of problem (0??) by orthogonal steps:

$$\bar{w}_k = -(c_k^2/\beta_k) q_k + s_k^2 \bar{w}_{k-2}^{(3)}, \quad \bar{w}_{k-2}^{(4)} = (s_k^2/\beta_k) q_k + c_k^2 \bar{w}_{k-2}^{(3)},$$

$$\bar{w}_k^{(3)} = s_k \bar{w}_k^{(2)} - c_k \bar{w}_k^{(2)}, \quad \bar{w}_{k-1}^{(3)} = c_k \bar{w}_{k-1}^{(2)} + s_k \bar{w}_k^{(2)},$$

$$x_{k-2} = x_{k-3} + \epsilon_k^{(3)} \bar{w}_{k-2}^{(4)}, \quad x_k = x_{k-2} + \mu_k \bar{w}_k^{(2)}.$$
Let $\bar{r}_k = \bar{b} - M^{-\frac{1}{2}} A M^{-\frac{1}{2}} \bar{x}_k = M^{-\frac{1}{2}} r_k$. Then $x_k = M^{1/2} \bar{x}_k$ is an acceptable solution of (0??) if the computed value of $\phi_k \approx \|\bar{r}_k\| = \|r_k\| M^{-1/2}$ is sufficiently small. If desirable, the corresponding residual norm can be obtained by $\|r_k\| = \sqrt{\bar{r}_k^T (M \bar{r}_k)}$.

We can now present our pseudocode in Algorithm 0??, The left and right reflections are implemented in Algorithm 0?? SymOrtho$(a, b)$, which is a stable form for computing $r = \sqrt{a^2 + b^2}$, $c = \frac{a}{r}$, and $s = \frac{b}{r}$. The complexity is at most 6 flops and a square root. Algorithm 0?? lists all steps of MINRES-QLP with preconditioning. For simplicity, $\bar{w}_k$ is written as $w_k$ for all relevant $k$. Also, the output $x$ solves $Ax \approx b$ but other outputs are associated with the preconditioned system.

4. KEY FORTRAN 90 DESIGN FEATURES

Our FORTRAN 90 package contains the following files for symmetric problems with their dependencies depicted in Figure 0??:

—minresqlpDataModule.f90: defines public constants for use in other modules
—minresqlpBlasModule.f90: packages some BLAS functions in a module
—minresqlpModule.f90: implements MINRES-QLP with preconditioning
—minresqlpTestModule.f90: illustrates how MINRES-QLP can call Aprod with a short fixed parameter list, even if it needs arbitrary other data
—minresqlpTestProgram.f90: contains the main driver program with unit tests.

In addition, we have the counterparts of these programs for Hermitian problems and they have the same filenames prefixed with the letter ‘z’.

In our FORTRAN 90 implementation, we use modules instead of COMMON blocks in FORTRAN 77 for the purpose of grouping programs units and data together, and controlling their availability to other program units. A module can use...
Algorithm 1: Pseudocode of preconditoned MINRES-QLP to solve $Ax \approx b$.  
In the right-justified comments, $A \equiv M^{-\frac{1}{2}} A M^{-\frac{1}{2}}$.

\textbf{input:} $A, b, \sigma, M$

\begin{enumerate}
  \item $z_0 = 0, \quad z_1 = b, \quad \text{Solve } Mq_1 = z_1, \quad \beta_1 = \sqrt{b^T q_1}$ [Initialize]
  \item $w_0 = w_{-1} = 0, \quad x_{-2} = x_{-1} = x_0 = 0$
  \item $c_0 = \sigma_0 = c_0, \quad c_0 = \sigma_0 = 0, \quad \phi_0 = \beta_1, \quad \tau_0 = \omega_0 = \chi_{-2} = \chi_{-1} = \chi_0 = 0$
  \item $\delta_1 = \gamma_1 = \gamma_0 = \eta_1 = \vartheta_1 = \vartheta_0 = \vartheta_1 = \mu_1 = \mu_0 = 0, \quad \alpha_0 = 0, \quad \kappa_0 = 1$
  \item $k = 0$

\end{enumerate}

\textbf{while} no stopping condition is satisfied do

\begin{enumerate}
  \item $k \leftarrow k + 1$
  \item $p_k = Ap_k - \sigma q_k, \quad \alpha_k = \frac{1}{\sigma} \bar{q}_k^T p_k$ [Precondioned Lanczos]
  \item $z_{k+1} = \frac{1}{\alpha_k} p_k - \frac{\alpha_k}{\beta_k} \bar{z}_k \bar{z}_{k-1}$
  \item Solve $Mq_{k+1} = z_{k+1}, \quad \beta_{k+1} = \sqrt{\bar{q}_{k+1}^T \bar{z}_{k+1}}$
  \item if $k = 1$ then $p_k = \|q_k \beta_{k+1}\| \text{ else } p_k = \|\beta_k, \alpha_k, \beta_{k+1}\|$
  \item $\delta_k^{(2)} = c_{k-1} \delta_k + s_{k-1} \alpha_k$ [Previous left reflection...]
  \item $\gamma_k = s_{k-1} \delta_k - c_{k-1} \alpha_k$ [on middle two entries of $T_k e_k$...]
  \item $\kappa_{k+1} = s_{k-1} \beta_{k+1}$ [produces first two entries in $T_{k+1} e_{k+1}$]
  \item $\delta_{k+1} = -c_{k-1} \beta_{k+1}$
  \item $c_{k+1}, s_{k+1}, \gamma_{k+1}^{(2)} \leftarrow \text{SymOrtho}(\gamma_k, \beta_{k+1})$ [Current left reflection]
  \item $c_{k+2}, s_{k+2}, \gamma_k^{(4)} \leftarrow \text{SymOrtho}(\gamma_k^{(2)}, \epsilon_k)$ [First right reflection]
  \item $r_k^{(2)} = s_k \delta_{k-1} - c_k \delta_k^{(2)}, \quad \gamma_k = -c_k \gamma_k^{(2)}, \quad \eta_k = s_k \gamma_k^{(2)}$
  \item $\psi_k^{(2)} = c_k \varphi_{k-1} + s_k \delta_k^{(2)}$
  \item $c_{k+3}, s_{k+3}, \gamma_{k+1}^{(5)} \leftarrow \text{SymOrtho}(\gamma_{k+1}^{(4)}, \delta_k^{(3)})$ [Second right reflection...]
  \item $\vartheta_k = s_k \gamma_k^{(3)} \quad \gamma_k = -c_k \gamma_k^{(3)}$ [to zero out $\delta_k^{(3)}$]
  \item $\tau_k = c_k \varphi_{k-1}$ [Last element of $t_k$]
  \item $\phi_k = s_k \varphi_{k-1}, \quad \psi_{k-1} = \varphi_{k-1} \|q_k \delta_{k+1}\|$ [Update $\|r_k\|, \|A r_{k-1}\|$
  \item if $k = 1$ then $\gamma_{\min} = \gamma_1 \text{ else } \gamma_{\min} \leftarrow \min \{\gamma_{\min}, \gamma_{\min}^{(2)} \gamma_{\min}^{(3)} \}$
  \item $A_k = \max \{|A_{k-1}, \rho_k, \gamma_{\min}^{(2)} \gamma_{\min}^{(3)} \}|$ [Update $||A||$
  \item $\omega_k = \|\omega_{k-1} \tau_k\|, \quad \kappa_k = A_k / \gamma_{\min}$ [Update $||A\omega_k||, \text{cond}(A)$]
  \item $w_k = -(c_k / \beta_k) \varphi_k + s_k \varphi_{k-1}$ [Update $w_{k-2}, w_{k-1}, w_k$
  \item $w_k^{(3)} = (s_k / \beta_k) \varphi_k + c_k \varphi_{k-1}$
  \item if $k > 2$ then $w_k^{(2)} = s_k \varphi_k - c_k \varphi_{k-1}, \quad w_k^{(4)} = c_k \varphi_k + s_k \varphi_{k-1}$
  \item if $k > 2$ then $\mu_k^{(2)} = (\tau_k - \eta_k \mu_k^{(4)} - \vartheta_k \mu_k^{(3)}) / \gamma_k^{(2)}$ [Update $\mu_{k-2}$]
  \item if $k > 1$ then $\mu_k^{(2)} = (\tau_k - \eta_k \mu_k^{(4)} - \vartheta_k \mu_k^{(3)}) / \gamma_k^{(2)}$ [Update $\mu_{k-1}$]
  \item if $\gamma_k^{(2)} \neq 0$ then $\mu_k = (\tau_k - \eta_k \mu_k^{(4)} - \vartheta_k \mu_k^{(3)}) / \gamma_k^{(2)}$ else $\mu_k = 0$ [Compute $\mu_k$]
  \item $\chi_{k-2} = x_{k-3} + \mu_{k-2} w_{k-2}$ [Update $x_{k-2}$]
  \item $x_k = x_{k-2} + \mu_{k-2} w_{k-2}$ [Compute $x_k$]
  \item $\chi_{k-2} = \|\chi_{k-3} \mu_{k-2}||$ [Update $||x_{k-2}||$
  \item $\chi_{k} = \|\chi_{k-2} \mu_{k-2}|$ [Compute $||x_{k}||$

\end{enumerate}

\textbf{output:} $x, \phi, \psi, \chi, A, \kappa, \omega$

Algorithm 2: Algorithm SymOrtho.

\begin{algorithm}
\begin{algorithmic}[1]
\State \textbf{input}: $a, b$
\If{$b = 0$} \State $s = 0$, $r = |a|$ \EndIf
\If{$a = 0$} \State $c = 1$ \Else $c = \text{sign}(a)$ \EndIf
\ElseIf{$a = 0$} \State $c = 0$, $s = \text{sign}(b)$, $r = |b|$ \EndIf
\ElseIf{$|b| > |a|$} \State $\tau = a/b$, $s = \text{sign}(b)/\sqrt{1 + \tau^2}$, $c = s\tau$, $r = b/s$ \EndIf
\ElseIf{$|a| > |b|$} \State $\tau = b/a$, $c = \text{sign}(a)/\sqrt{1 + \tau^2}$, $s = ct$, $r = a/c$ \EndIf
\end{algorithmic}
\end{algorithm}

\textbf{output}: $c, s, r$

public data and subroutines from other modules (by declaring an \textit{interface} block), share its own public data and subroutines with other program units, and hide its own private data and subroutines from being used by other program units.

In minresqlpModule.f90, we define a public subroutine \texttt{MINRESQLP}—this is where we implement MINRES-QLP in Algorithm 0???. The second and the third input arguments of this subroutine, \texttt{Aprod} and \texttt{Msolve}, are external user-defined private subroutines. The subroutine \texttt{Aprod} defines the matrix $A$ as an operator. For a given vector $x$, the FORTRAN statement \texttt{call Aprod(n, x, y)} must return the product $y = Ax$ without altering the vector $x$. The subroutine \texttt{Msolve} is optional and it defines a symmetric positive definite matrix as an operator $M$ that serves as a preconditioner. For a given vector $y$, the FORTRAN statement \texttt{call Msolve(n, y, x)} must solve the linear system $Mx = y$ without altering the vector $y$. To provide the compiler the necessary information about these private subroutines defined in \texttt{minresqlpTestModule}, an \textit{interface} block in subroutine \texttt{MINRESQLP} is declared, which essentially replicates the headers of \texttt{Aprod} and \texttt{Msolve} in \texttt{minresqlpTestModule}.

A public routine \texttt{minresqlptest}, also defined in module \texttt{minresqlpTestModule}, calls \texttt{MINRESQLP} with \texttt{Aprod} and \texttt{Msolve} passed in to \texttt{MINRESQLP}.

We mark all data variables in \texttt{minresqlpTestModule} used for defining \texttt{Aprod} and \texttt{Msolve} private so that they are accessible to all the subroutines in the module but not outside.

To summarize, we have described and provided a pattern that allows MINRESQLP users to solve different problems by simply editing \texttt{minresTestModule} (and possibly the main program \texttt{minresTestProgram}, which calls \texttt{minresqlptest}). They do not need to change \texttt{MINRESQLP} as long as the header of subroutines \texttt{Aprod} and \texttt{Msolve} stay the same in \texttt{minresTestModule}.

In addition, our design spares users from writing subroutines for \textit{reverse communication}, which enables the development of iterative methods without the a priori knowledge of users’ problem data $A$ and $M$. It returns control to the calling program every time before \texttt{Aprod} or \texttt{Msolve} is invoked. While reverse communication is widely used in scientific computing with FORTRAN 77, the subroutine calls represent an overhead and the resulting code usually appears formidable and often unrecognizable from the original pseudocode; see [?] and [?] for two exam-
ples of CG and numerical integration coded in FORTRAN 77 and FORTRAN 90 respectively. Our MINRES-QLP implementation achieves the purpose of reverse communication while preserving code readability and thus maintainability. The FORTRAN 90 module structure allows user’s $Ax$ products and $Mx = y$ solve to be implemented outside MINRES-QLP in the same way that MATLAB’s function handles operate.

Finally, unit testing is an important software development strategy that cannot be overemphasized and this may be especially true in the scientific computing communities. Unit testing usually consists of multiple small and fast but very specific and illuminating test cases that check if the code behaves as designed. Software development is incremental in nature and errors (also known as bugs) are often found over time. Adding new functionalities or fixing errors often result in unknowingly breaking the code for some earlier successful test cases. It is therefore critical to continuously expand the test cases, and assure that all unit tests are executed with expected results every time after a key program unit is updated.

In our development of MINRES-QLP, we have created a suite of 21 test cases in minresqlpTestProgram. By default, the test program outputs test results to MINRESQLP.txt. If users ever have the need to modify the code in the subroutine MINRESQLP, they could run these test cases, grep (a Unix command) the word “appear” in the test output, check if all tests are reported to be successful, and let it be an acceptance criteria for the code change. For more sophisticated unit testing frameworks employed in large-scale scientific software development, see [?].

For more details on the implementation, documentation, and numerical examples, we refer readers to [?].