Adaptive Projection Subspace Dimension for the Thick-Restart Lanczos Method

ICHITARO YAMAZAKI and ZHAOJUN BAI
University of California, Davis
and
HORST SIMON, LIN-WANG WANG, and KESHENG WU
Lawrence Berkeley National Laboratory

The Thick-Restart Lanczos (TRLan) method is an effective method for solving large-scale Hermitian eigenvalue problems. The performance of the method strongly depends on the dimension of the projection subspace used at each restart. In this article, we propose an objective function to quantify the effectiveness of the selection of subspace dimension, and then introduce an adaptive scheme to dynamically select the dimension to optimize the performance. We have developed an open-source software package *a*–TRLan to include this adaptive scheme in the TRLan method. When applied to calculate the electronic structure of quantum dots, *a*–TRLan runs up to 2.3x faster than a state-of-the-art preconditioned conjugate gradient eigensolver.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*Eigenvalues and eigenvectors (direct and iterative methods)*

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Adaptive subspace dimension, Lanczos, thick-restart, electronic structure calculation

This work was supported in part by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Z. Bai was also supported in part by NSF grants DMS-0611548 and OCI-0749217 and DOE grant DE-FC02-06ER25794. This research used resources from the National Energy Research Scientific Computing Center, which is supported by the Office of Energy Research of the U.S. Department of Energy.
Authors' addresses: I. Yamazaki and Z. Bai, Department of Computer Science, University of California, One Shields Avenue, Davis, CA 95616; email: {yamazaki, bai}@cs.ucdavis.edu. H. Simon, L.-W. Wang, and K. Wu, Lawrence Berkeley National Laboratory, One Cyclotron Road, Berkeley, CA 94720; email: {hdsimon, lwwang, kwu}@lbl.gov.

© 2010 ACM 0098-3500/2010/09-ART27 $10.00 DOI: 10.1145/1824801.1824805. http://doi.acm.org/10.1145/1824801.1824805.

ACM Reference Format:
Yamazaki I., Bai, Z., Simon, H., Wang, L.-W., and Wu, K. 2010. Adaptive projection subspace dimension for the thick-restart Lanczos method. ACM Trans. Math. Softw. 37, 3, Article 27 (September 2010), 18 pages. DOI = 10.1145/1824801.1824805.
http://doi.acm.org/10.1145/1824801.1824805.

1. INTRODUCTION

The Lanczos method [Lanczos 1950] for solving large-scale Hermitian eigenvalue problems computes a new orthonormal basis vector of a projection subspace at each iteration. Computational and memory costs increase rapidly as the iteration proceeds. To reduce the costs, the method is typically restarted after the projection subspace of a fixed dimension is computed [Sorensen 1992; Wu and Simon 2000a]. The performance of the restarted method strongly depends on the dimension of the projection subspace. If it is selected to be too small, the method suffers from slow convergence. On the other hand, if it is too large, the computational and memory costs become expensive and the overall performance suffers. In order to achieve an optimal performance, it is necessary to select a proper subspace dimension that balances the costs and the convergence rate. To demonstrate this delicate task, let us examine the performance of the Thick-Restart Lanczos (TRLan) method [Wu and Simon 2000b]. Figure 1 shows the numbers of matrix-vector products (in thousands) and CPU times (in seconds) of the TRLan method to compute the smallest 20 eigenvalues of a 10000×10000 diagonal matrix $A = \mathrm{diag}(1, 2^2, 3^2, \ldots, 10000^2)$ with respect to different subspace dimensions. As we can see, a larger subspace dimension improves the convergence rate (i.e., TRLan converges with a smaller number of matrix-vector products). However, as the subspace dimension becomes too large, the CPU time starts to increase dramatically.

In order to free users from having to select an appropriate subspace dimension, in this article, we propose an adaptive scheme to dynamically select the subspace dimension in conjunction with the TRLan method. We first distinguish between a prescribed maximum dimension of the subspace and the dimension of the subspace actually used at each restart, and then introduce an objective function to quantify the effectiveness of a subspace dimension in balancing the cost and the convergence rate. The subspace dimension is then dynamically determined to optimize the objective function. We refer to the TRLan method with this adaptive scheme to select the subspace dimension as an *a–TRLan method*.

In the original Fortran 90 implementation of the TRLan method, the dimension of the projection subspace is static and prescribed for solving real symmetric eigenvalue problems [Wu and Simon 2000b]. We have rewritten the TRLan method in C and extended it to include the adaptive scheme described in this article and to solve complex Hermitian eigenvalue problems [Yamazaki et al. 2008]. As with the original implementation, the Message Passing Interface (MPI) is used on distributed memory systems.

Numerical results of *a*–TRLan for solvng synthetic and realistic problems from different applications demonstrate that *a*–TRLan not only automates the

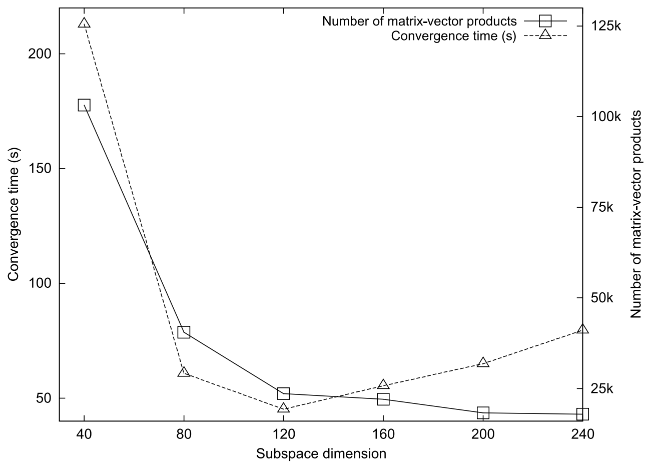

Fig. 1. Performance of TRLan with different subspace dimension.

selection of the subspace dimension, but also improves the performance of TRLan with optimal static subspace dimension. We have integrated a–TRLan into the Parallel Energy Scan (PESCAN) code [Wang et al. 2008]. PESCAN has been successfully used to calculate the electronic structures of semiconductor quantum dots [Canning et al. 2000; Wang and Zunger 1994] and for other applications [Li and Wang 2004; Schrier and Wang 2006]. The state-of-the-art eigensolver in the PESCAN is based on the Preconditioned Conjugate Gradient (PCG) method [Payne et al. 1992]. Numerical results show that a–TRLan is significantly faster than the PCG-based eigensolver with speedups of up to 2.3 for computing as few as 30 eigenpairs of interest.

2. THICK-RESTART LANCZOS METHOD

The Lanczos method [Lanczos 1950] is an effective method for computing a few exterior eigenvalues λ and their corresponding eigenvectors v of a Hermitian matrix A

$$Av = \lambda v. \tag{1}$$

Given a starting vector q, the Lanczos method first computes orthonormal basis vectors $q_1, q_2, \ldots, q_{i+1}$ of a Krylov subspace

$$\mathcal{K}_{i+1}(q, A) \equiv \text{span}\{q, Aq, A^2q, \ldots, A^iq\}.$$

These basis vectors satisfy the relation

$$AQ_i = Q_iT_i + \beta_i q_{i+1} e_i^H, \tag{2}$$

where $Q_i = [q_1, q_2, \ldots, q_i]$, $\beta_i = q_{i+1}^H A q_i$, e_i is the ith column of the identity matrix of order i, $T_i = Q_i^H A Q_i$ is a Rayleigh-Ritz projection of A onto $\mathcal{K}_i(A, q)$, and the superscript ^{H} indicates the conjugate transpose. An approximate eigenpair $(\theta, x = Q_i y)$ of A is computed from an eigenpair (θ, y) of T_i. These approximate eigenvalue θ and eigenvector x are referred to as Ritz value and Ritz

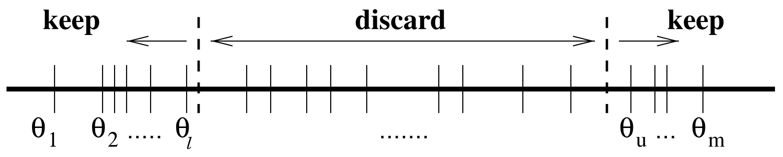

Fig. 2. Ritz values to be kept at restart.

vector, respectively. The accuracy of the Ritz pair (θ, x) is measured by the residual norm

$$\|r\|_2 = \|Ax - \theta x\|_2 = \|(AQ_i - Q_iT_i)y\|_2 = \beta_i\|q_{i+1}e_i^H y\|_2 = \beta_i|y(i)|, \tag{3}$$

where $y(i)$ is the ith element of y. The Ritz pair (θ, x) is converged if the residual norm defined in (3) is less than a prescribed threshold. It is well known that Ritz values typically converge to exterior eigenvalues of A with a subspace dimension i that is much smaller than the dimension n of A [Parlett 1998; Saad 1993].

A key feature that distinguishes the Lanczos method from other subspace projection methods is that T_i of (2) is symmetric tridiagonal:

$$T_i = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{i-2} & \alpha_{i-1} & \beta_{i-1} \\ & & & \beta_{i-1} & \alpha_i \end{pmatrix}.$$

Consequently, it leads to the following simple three-term recurrence:

$$\beta_i q_{i+1} = Aq_i - \alpha_i q_i - \beta_{i-1}q_{i-1}. \tag{4}$$

The new basis vector q_{i+1} can be computed by orthonormalizing the vector Aq_i against only two preceding basis vectors, q_{i-1} and q_i. Unfortunately, in finite precision arithmetic, when the new basis vector q_{i+1} is computed by (4), the orthogonality among the basis vectors is lost even after a small number of iterations. To maintain orthogonality, the new basis vector q_{i+1} is reorthogonalized against all the previous vectors $q_1, q_2, \ldots, q_i$. This reorthogonalization process is typically carried out using a Gram-Schmidt procedure [Parlett 1998; Saad 1993]. As the basis size $i + 1$ grows, this process becomes computationally expensive. Furthermore, all the basis vectors Q_i need to be stored in memory.

To reduce the costs of computing a large subspace, the iteration is restarted after a fixed number of the basis vectors are computed. Since the Ritz values first converge to the exterior eigenvalues of A, TRLan selects two indices ℓ and u to indicate those Ritz values to be kept at both ends of the spectrum (see Figure 2). The corresponding kept Ritz vectors are denoted by

$$\widehat{Q}_k = [\widehat{q}_1, \widehat{q}_2, \ldots, \widehat{q}_k] = Q_m Y_k, \tag{5}$$

where m is the dimension of the subspace,

$$Y_k = [y_1, y_2, \ldots, y_\ell, \; y_u, y_{u+1}, \ldots, y_m], \tag{6}$$

and y_i is the eigenvector of T_m corresponding to θ_i. TRLan sets these Ritz vectors $\widehat{Q}_k$ as the first k basis vectors at the restart and q_{m+1} as the $(k+1)$th basis vector (i.e., $\widehat{q}_{k+1} = q_{m+1}$).[1] To compute the $(k+2)$th basis vector $\widehat{q}_{k+2}$, TRLan computes $A\widehat{q}_{k+1}$ and orthonormalizes it against the previous $k+1$ basis vectors as follows.

$$\widehat{\beta}_{k+1}\widehat{q}_{k+2} = A\widehat{q}_{k+1} - \widehat{Q}_k(\widehat{Q}_k^H A\widehat{q}_{k+1}) - \widehat{q}_{k+1}(\widehat{q}_{k+1}^H A\widehat{q}_{k+1}). \tag{7}$$

Note that $A\widehat{Q}_k$ satisfies the relation

$$A\widehat{Q}_k = \widehat{Q}_k D_k + \beta_m \widehat{q}_{k+1} s^H,$$

where D_k is the $k \times k$ diagonal matrix whose diagonal elements are the kept Ritz values, and $s = Y_k^H e_m$. Hence, the coefficients $\widehat{Q}_k^H A\widehat{q}_{k+1}$ in (7) can be computed efficiently:

$$\begin{aligned}\widehat{Q}_k^H A\widehat{q}_{k+1} &= (A\widehat{Q}_k)^H \widehat{q}_{k+1} = (\widehat{Q}_k D_k + \beta_m \widehat{q}_{k+1} s^H)^H \widehat{q}_{k+1} \\ &= D_k Y_k^H (Q_m^H q_{m+1}) + \beta_m s(\widehat{q}_{k+1}^H \widehat{q}_{k+1}) = \beta_m s\,.\end{aligned}$$

In general, at the ith iteration after the restart, the new basis vector $\widehat{q}_{k+i+1}$ satisfies the relation

$$A\widehat{Q}_{k+i} = \widehat{Q}_{k+i}\widehat{T}_{k+i} + \widehat{\beta}_{k+i}\widehat{q}_{k+i+1} e_{k+i}^H,$$

where $\widehat{T}_{k+i} = \widehat{Q}_{k+i}^H A\widehat{Q}_{k+i}$ is of the form

$$\widehat{T}_{k+i} = \begin{pmatrix} D_k & \beta_m s & & & & \\ \beta_m s^H & \widehat{\alpha}_{k+1} & \widehat{\beta}_{k+1} & & & \\ & \widehat{\beta}_{k+1} & \widehat{\alpha}_{k+2} & \widehat{\beta}_{k+2} & & \\ & & \ddots & \ddots & \ddots & \\ & & & \widehat{\beta}_{k+i-2} & \widehat{\alpha}_{k+i-1} & \widehat{\beta}_{k+i-1} \\ & & & & \widehat{\beta}_{k+i-1} & \widehat{\alpha}_{k+i} \end{pmatrix}.$$

Note that the three-term recurrence is not valid only for computing the $(k+2)$th basis vector and is resumed afterward. Figure 3 shows the pseudocode of the TRLan algorithm, where $m_{j+1} = m$ for all j at step 3.e. A detailed description of the TRLan method can be found in Wu and Simon [2000a].

Besides the TRLan method, there are several other restarting schemes. One can restart the iteration with a new starting vector, such as a linear combination of the current approximate eigenvectors. Unfortunately, this simple approach loses a significant amount of information at restart and results in slow convergence. The implicitly restart Lanczos method [Calvetti et al. 1994] keeps a fixed number of vectors at restart which approximately span a subspace containing the desired Ritz vectors by filtering out the unwanted ones. The TRLan method allows more explicit control over the selections of Ritz vectors.

[1]The ith basis vector $\widehat{q}_i$ computed after the restart is distinguished from the ith basis vector q_i computed before the restart by the hat over it.

```
set $q_1 = q/\|q\|_2$, $k = 0$, and $m = m_1$.
for $j = 1, 2, 3, \ldots$
    1. Initialization.
        b.  $p = Aq_{k+1}$
        c.  $\alpha_{k+1} = q_{k+1}^H p$
        d.  $p = p - \alpha_{k+1} q_{k+1} - \sum_{i=1}^{k} \beta_i q_i$
        e.  $\beta_{k+1} = \|p\|_2$
        f.  $q_{k+2} = p/\beta_{k+1}$
    2. The $j$-th restart-loop.
        a.  for $i = k+2, k+3, \ldots, m_j$
        b.      $p = Aq_i$
        c.      $\alpha_i = q_i^H p$
        d.      $p = p - \alpha_i q_i - \beta_{i-1} q_{i-1}$
        e.      reorthogonalize $p$ if necessary.[2]
        f.      $\beta_i = \|p\|_2$
        g.      $q_{i+1} = p/\beta_i$
        h.  end for
    3. The $j$-th restart.
        a.  compute all $\theta_i$ and $y_i(m_j)$ of $T_{m_j}$ and compute (3).
        b.  if stopping criterion is satisfied then
        c.      compute desired Ritz vectors and exit.
        d.  else restart:
        e.      select $(\ell_{j+1}, u_{j+1}, m_{j+1})$.
        f.      set $k = \ell_{j+1} + m_j - u_{j+1} + 1$ and $m = m_{j+1}$.
        g.      compute eigenvectors $Y_k$ of (6).
        h.      compute Ritz vectors $\widehat{Q}_k$ of (5).
        i.      set $\{q_1, q_2, \ldots, q_k\} = \widehat{Q}_k$ and $q_{k+1} = q_{m+1}$.
        j.      set $\alpha_i = \theta_{\pi_i}$ and $\beta_i = \beta_m y_{\pi_i}(m)$, for $i = 1, \ldots, k$,
                where $(\pi_1, \pi_2, \ldots, \pi_k) = (1, \ldots, \ell, u, u+1, \ldots, m)$.
        k.  end if
end for
```

Fig. 3. Pseudocode of the TRLan algorithm.

3. ADAPTIVE SCHEME TO DETERMINE PROJECTION SUBSPACE DIMENSION

At step 3.e of the TRLan pseudocode in Figure 3, a triplet $(\ell_{j+1}, u_{j+1}, m_{j+1})$ is selected to specify the Ritz vectors to be kept and the dimension of the projection subspace to be used at the next restart. In the original implementation of TRLan [Wu and Simon 2000b], the Ritz vectors are selected to maximize the convergence rate, while the subspace dimension m_{j+1} is fixed to be the user-selected maximum dimension $m_{\max}$ (see Wu and Simon [2000b, Section 6.1] for the discussion on how to select $m_{\max}$). As we have discussed in Section 1, it is a nontrivial task to select an optimal value of m_{j+1} since it depends on a number of factors, such as the available memory and the distribution of the eigenvalues. In this section, we introduce an adaptive scheme to determine m_{j+1}. We will first examine the expected convergence rate and the associated computational cost over the next restart-loop, and then introduce an objective function to measure the effectiveness of the triplet $(\ell_{j+1}, u_{j+1}, m_{j+1})$ in terms of balancing the cost and the convergence rate.

3.1 Convergence Factor

Let us examine the convergence rate of the $(\ell + 1)$th smallest Ritz value over the $(j + 1)$-th restart-loop, where the Ritz values $\theta_1, \dots, \theta_\ell$ and $\theta_u, \dots, \theta_{m_j}$ are assumed to have converged. We use $\|r_j\|_2$ to denote the residual norm (3) of the Ritz pair $(\theta_{\ell+1}, x_{\ell+1})$ at the jth restart, and use ω_j to denote the reciprocal of the reduction factor of $\|r_j\|_2$ over the $(j + 1)$th restart-loop as follows.

$$\|r_{j+1}\|_2 = \frac{1}{\omega_j}\|r_j\|_2 .$$

According to the analysis of Morgan [1996], after the $m - k$ Lanczos iterations, ω_j is approximately given by

$$\omega_j \simeq \mathcal{C}_{m-k}(1 + 2\gamma),$$

where $\mathcal{C}_{m-k}$ is the Chebyshev polynomial of degree $m - k$, and γ is the spectral gap ratio defined as

$$\gamma = \frac{\lambda_{\ell+2} - \lambda_{\ell+1}}{\lambda_{n-(m_j-u+1)} - \lambda_{\ell+2}}. \tag{8}$$

Note that ℓ and $(m_j - u + 1)$ are the numbers of the smallest and largest converged Ritz pairs, respectively, and $0 \le \ell < u \le m_j + 1$. If $\ell = 0$ or $u = m_j + 1$, then the smallest or largest Ritz values have not yet converged, respectively. We also note that $m > k = \ell + (m_j - u + 1)$, where m is a candidate dimension of the next projection subspace, while m_j is the subspace dimension used at the jth restart.

For large-scale eigenvalue problems of practical interest, it is typical that $\gamma \ll 1$. Hence, we use the following estimate of ω_j when $0 < \gamma \ll 1$.

$$\omega_j \simeq \cosh(2(m-k)\sqrt{\gamma}) \simeq 2(m-k)\sqrt{\gamma} . \tag{9}$$

For the approximations of Chebyshev polynomials, for example, see Demmel [1997, Lemma 6.7].

In practice, the exact eigenvalues of A are not available. Hence, we replace the eigenvalues λ_i in the definition of the gap ratio (8) with the corresponding computed Ritz values and use the following *effective* gap ratio γ_e to compute ω_j.

$$\gamma_e = \frac{\theta_{\ell+2} - \theta_{\ell+1}}{\theta_{u-1} - \theta_{\ell+2}} . \tag{10}$$

To measure the convergence rate of the $(u - 1)$th Ritz value using ω_j, the effective gap ratio (10) needs to be changed to

$$\gamma_e = \frac{\theta_{u-1} - \theta_{u-2}}{\theta_{u-2} - \theta_{\ell+1}} .$$

For the rest of this article, we focus on the convergence rate of the $(\ell + 1)$th Ritz value for computing a few smallest eigenvalues.

3.2 Computational and Memory Costs

Beside the matrix-vector product (step 2.b in Figure 3), the dominant computational costs of the TRLan method are as follows.

(1) Reorthogonalization (step 2.e in Figure 3): When a new basis vector q_{i+1} is reorthogonalized against all the previous basis vectors using the Gram-Schmidt procedure, it requires approximately $4ni$ floating-point operations (flops). For simplicity, we consider the full reorthogonalization.[2] The aggregated cost of the reorthogonalization is approximately given by

$$\sum_{i=k}^{m-1} 4ni = 2n(m-k)(k+m-1) \text{ flops},$$

where $k = \ell + m_j - u + 1$.

(2) Ritz vector computation (step 3.h in Figure 3): The cost of computing the Ritz vectors $\widehat{Q}_k = Q_m Y_k$ requires approximately

$$2nmk \text{ flops}.$$

Therefore, aside from the flops of the matrix-vector products, the total number of flops required for the next restart-loop is approximately

$$2n(m-k)(k+m-1) + 2nmk.$$

On a modern computer, the number of flops may not be an accurate measure of the expected running time of a program because the number of memory references and the memory access pattern may dominate. For example, the average time spent for a flop in the sparse matrix-vector product could be significantly greater than that in the Ritz vector computation due to the irregular data access of the sparse matrix-vector product. To incorporate this factor, we use the following formula to model the expected running time of the next restart-loop:

$$\alpha_1(2n(m-k)(k+m-1)) + \alpha_2(2nmk) + \alpha_3(m-k), \tag{11}$$

where α_1 and α_2 are the average time spent per flop in the reorthogonalization and Ritz vector computation, respectively, and α_3 is the average time for a sparse matrix-vector product. The parameters α_1, α_2, and α_3 are computed based on the measured times from the previous iterations.

3.3 Objective Function

Based on the reduction factor (9) and the computational cost (11), we define the following objective function to model the effectiveness of the triplet (ℓ, u, m):

$$f(\ell, u, m) = \frac{(m-k)\sqrt{\gamma_e}}{n(\alpha_1(m-k)(k+m-1) + \alpha_2 mk) + \alpha_3(m-k)}, \tag{12}$$

[2]Both TRLan and α–TRLan implement a selected reorthogonalization scheme as described in Parlett [1998, Section 6.9]. In practice, we have observed that most of the new basis vectors need to be fully reorthogonalized.

where $k = \ell + m_j - u + 1$. An optimal triplet $(\ell_{\text{opt}}, u_{\text{opt}}, m_{\text{opt}})$ should balance the computational cost and the convergence rate over the next restart-loop, and hence it is given by $(\ell_{\text{opt}}, u_{\text{opt}}, m_{\text{opt}}) = \arg\max f(\ell, u, m)$.

3.4 Practical Issues

Let c_ℓ and c_u be the numbers of the smallest and largest Ritz values satisfying a prescribed convergence criterion, respectively. If only the converged Ritz pairs are kept as discussed in Section 3.1, then the two indices ℓ_{j+1} and u_{j+1} to specify the kept Ritz pairs are given by

$$\ell_{j+1} = c_\ell \quad \text{and} \quad u_{j+1} = m_j - c_u + 1, \tag{13}$$

while the next subspace dimension is given by $m_{j+1} = \arg\max f(\ell_{j+1}, u_{j+1}, m)$ with $m > k$.

However, in practice, the convergence rate of the target Ritz pair $(\theta_{c_\ell+1}, x_{c_\ell+1})$ can be improved by keeping the Ritz pairs around the target even though they have not yet converged. This is because the kept Ritz vectors approximately deflate the spectrum of the eigenvectors around the target and increase the separation between them. Thus, instead of (13), we enforce the following constraints on the indices ℓ and u:

$$c_\ell + 1 \le \ell \quad \text{and} \quad u \le m_j + 1 \tag{14}$$

(initially, $c_\ell = 0$). In addition, since interior Ritz values are slow to converge, we enforce a minimum gap g_j between the indices ℓ and u to avoid keeping the interior Ritz values that have not converged at all:

$$g_j = \nu \cdot (m_j - c_\ell), \tag{15}$$

where $m_j - c_\ell$ is the maximum possible gap, and ν is a relaxation factor, $0 \le \nu \le 1$. In the case of $\nu = 1$, only the smallest converged Ritz pairs are kept, namely $\ell = c_\ell$ and $u = m_j + 1$. As the value of ν decreases, more Ritz pairs are allowed to be kept. The effect of ν will be discussed in Section 3.5.

Combining the constraints (14) and (15), we arrive at the following ranges of the indices ℓ and u.

$$\begin{aligned} c_\ell + 1 \le\ & \ell \le m_j + 1 - g_j\,, \\ \ell + g_j \le\ & u \le m_j + 1\,. \end{aligned} \tag{16}$$

The corresponding range for the subspace dimension m is

$$\ell + m_j - u + 2 \le m \le m_{\max}. \tag{17}$$

From the triplets (ℓ, u, m) satisfying (16) and (17), the optimal triplet $(\ell_{\text{opt}}, u_{\text{opt}}, m_{\text{opt}})$ is searched for based on a search algorithm shown in Figure 4, where ℓ_ℓ is a lower bound of ℓ and u_u is an upper bound of u (i.e., $\ell_\ell = c_\ell + 1$ and $u_u = m_j+1$). In practice, we found that when computing n_d smallest eigenvalues, the convergence rate can be improved by enforcing $\ell_\ell = n_d$ and $u_u = m_j$ such that at least n_d smallest Ritz values and one largest Ritz value were kept at restarts. Hence, these bounds are used in our implementation. We also found that when the maximum subspace dimension $m_{\max}$ is too small, the maximum subspace dimension is selected at every restart (i.e., $m_{j+1} = m_{\max}$ for all j). In such a case,

```
1.  Set $(\ell_{\rm opt}, u_{\rm opt}, m_{\rm opt}) = (\ell_\ell, \ell_\ell + g_j, m_j - g_j + 1)$
2.  for $\ell = \ell_\ell, \ell_\ell + 1, \ldots, m_j + 1 - g_j$
3.      for $u = \ell + g_j, \ell + g_j + 1 \ldots, u_u$
4.          $k = \ell + m_j - u + 1$
5.          for $m = k + 1, k + 2, \ldots, m_{\max}$
6.              if  $f(\ell, u, m) > f(\ell_{\rm opt}, u_{\rm opt}, m_{\rm opt})$ then
7.                  $(\ell_{\rm opt}, u_{\rm opt}, m_{\rm opt}) = (\ell, u, m)$
8.              end if
9.          end for
10.     end for
11  end for
```

Fig. 4. Pseudocode to search for $(\ell_{\rm opt}, u_{\rm opt}, m_{\rm opt})$.

the convergence rate is often improved by maximizing the reduction factor (9) than by trying to balance the cost with the convergence rate using (12). Hence, in our implementation, when the maximum subspace dimension is selected at a restart, the indices ℓ and u are recomputed to maximize

$$g(\ell, u) = (m_{\max} - k)\sqrt{\gamma_e}. \tag{18}$$

In summary, the proposed *a*–TRLan method uses the computed triplet $(\ell_{\rm opt}, u_{\rm opt}, m_{\rm opt})$ to replace the triplet $(\ell_{j+1}, u_{j+1}, m_{j+1})$ at the step 3.e of the pseudocode in Figure 3, while the initial subspace dimension is set to be $m_1 = \min(2n_d, m_{\max})$. The cost of selecting the triplet is $O(m_{\max}^3)$. In comparison to the total cost (11), it is insignificant since $m_{\max}$ is typically much smaller than n. Note that since the Ritz pairs that have not converged are now kept, the effective gap ratio (10) needs be replaced with

$$\gamma_e = \frac{\theta_{\ell+1} - \theta_{c_\ell+1}}{\theta_{u-1} - \theta_{\ell+1}}. \tag{19}$$

We note that TRLan selects the indices ℓ_{j+1} and u_{j+1} to maximize the convergence rate measured by (18), while the subspace dimension is fixed (i.e., $m_{j+1} = m_{\max}$ for all j). A similar restart scheme with a fixed subspace dimension was used for the thick-restarted Davidson method [Stathopoulos et al. 1998]. A different adaptive scheme to determine the projection subspace dimension for the Davidson method was studied in Crouzeix et al. [1994], where the iteration is restarted as soon as the product of the computational cost of a single iteration and the local convergence rate of the residual norm (i.e., $\|r_j\|_2/\|r_{j-1}\|_2$) grows significantly. Our adaptive scheme for *a*–TRLan, on the other hand, attempts to optimize the performance over the next restart-loop.

3.5 Heuristic for the Relaxation Factor ν

We now discuss the effect of the relaxation factor ν of (15) on the performance of *a*–TRLan. Figure 5 shows the CPU time of *a*–TRLan using $m_{\max} = 1000$ and different ν to compute $n_d = 100$ smallest eigenvalues of diagonal matrices $A_1 = \mathrm{diag}(1, 2, 3, \ldots, n)$ and $A_3 = \mathrm{diag}(1, 2^3, 3^3, \ldots, n^3)$ with $n = 10000$. The CPU times are normalized by that of $\nu = 0.1$. The figure clearly indicates the impact of ν on the performance of *a*–TRLan. It also shows that optimal performance

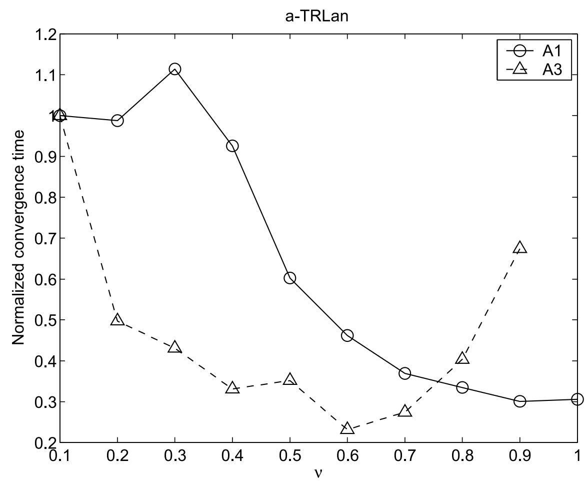

Fig. 5. Effect of relaxation factor ν on performance of a–TRLan.

of a–TRLan is achieved with different ν for A_1 and A_3. In the original TRLan implementation, it was fixed at $\nu = 0.4$.

To eliminate the need of a user to search for an optimal ν, we propose a scheme to dynamically adjust the relaxation factor ν based on the observed convergence rate. Specifically, we select the relaxation factor ν_j at the jth restart by considering the factor $\|r_j\|_2/\|r_{j-1}\|_2$ of the $(j-1)$th target Ritz pair $(\theta_{c_\ell+1}, x_{c_\ell+1})$, where $\|r_j\|_2$ is the residual norm of the target at the jth restart. Then, we define an *observed* gap ratio γ_o over the jth restart-loop as

$$\gamma_o = \left(\frac{\operatorname{arccosh} \frac{\|r_{j-1}\|_2}{\|r_j\|_2}}{2(m_j - k_j)} \right)^2, \tag{20}$$

where $k_j = \ell_j + m_j - u_j + 1$. Recall that the gap ratio γ was previously used in (9) to measure the expected convergence ratio.

A *desired* gap ratio γ_d which achieves good performance of a–TRLan is one which ensures that the target Ritz pair converges within two restart-loops:

$$\frac{\tau \|A\|_2}{\|r_j\|_2} = \frac{1}{\cosh(4\bar{m}\sqrt{\gamma_d})},$$

where τ is a required accuracy of the converged Ritz pairs (θ, x), that is, $\|Ax - \theta x\|_2 \leq \tau \|A\|_2$, $\|A\|_2$ is approximated by the largest absolute value of all the converged Ritz values, and $\bar{m}$ is the average dimension of the projection subspaces used at previous restarts. Thus, γ_d is computed as

$$\gamma_d = \left(\frac{\operatorname{arccosh}\left(\frac{\|r_{j-1}\|_2}{\tau \|A\|_2} \right)}{4\bar{m}} \right)^2. \tag{21}$$

When $\gamma_o < \gamma_d$, it indicates slow convergence. In this case, we attempt to improve the solution convergence for the next restart-loop by selecting a smaller value of ν_j and allowing more Ritz vectors to be kept. Otherwise, a larger value of ν_j is selected to reduce the computational cost. To automatically adjust the relaxation factor ν_j, we introduce the heuristic

$$\nu_j = \nu_\ell + \frac{2}{\pi}(1 - \nu_\ell)\arctan\left(\frac{\gamma_o}{\gamma_d}\right), \tag{22}$$

where ν_ℓ is a lower bound on ν_j, $0 \le \nu_\ell \le 1$. A good default lower bound of ν_ℓ is found to be 0.7.

We note that when the targetted residual norm did not decrease after the $m_j - k_j$ iterations, namely $\|r_{j-1}\|_2 \ge \|r_j\|_2$, the observed gap ratio γ_o of (20) is not defined. In this case, we use the default value $\nu_j = 0.7$. When a smaller solution accuracy τ is required, the desired gap ratio γ_d becomes larger, and a smaller relaxation factor ν_j is selected. Hence, more Ritz vectors are allowed to be kept for faster convergence.

4. NUMERICAL EXPERIMENTS

In this section, we present numerical results to compare the performance of the TRLan and *a*–TRLan methods. These methods are implemented in C and included in the open-source package *a*–TRLan [Yamazaki et al. 2008]. For all of our experiments, we used a vector of all ones as the initial vector q of the Lanczos iteration. A computed eigenpair (θ, x) is considered to be converged when its relative residual norm is smaller than a prescribed threshold τ, that is, $\|Ax - \theta x\|_2 \le \tau\|A\|_2$, where $\|A\|_2$ is approximated by the largest absolute value of the computed eigenvalues.

4.1 Synthetic Examples

Let us first present numerical results of some synthetic eigenvalue problems to illustrate the essential properties of the *a*–TRLan method. These numerical experiments were conducted on an HP Itanium2 workstation with a 1.5 GHz CPU and 2GB of RAM. The codes were compiled using the `icc` compiler (version 9.0) and the optimization flag `-O3`, and linked to the BLAS and LAPACK libraries in the Intel Math Kernel Library (version 7.2.1).[3] For the convergence criterion, we used $\tau = 10^{-13}$ for all the synthetic problems.

Example 1. We computed $n_d = 100$ smallest eigenvalues of diagonal matrices $A_1(n) = \mathrm{diag}(1, 2, \ldots, n)$ and $A_3(n) = \mathrm{diag}(1^3, 2^3, \ldots, n^3)$ with $n = 10000$. Figure 6 shows that the subspace dimension m_{j+1} is dynamically selected at the *j*th restart. As the iteration proceeds, the number n_c of converged eigenpairs increases, and the number k_j of kept Ritz pairs and the subspace dimension m_{j+1} also increase accordingly.

Figure 7 compares the elapsed CPU time required by TRLan and *a*–TRLan using different prescribed $m_{\max}$. The figure shows that the performance of *a*–TRLan is essentially independent of $m_{\max}$, while the performance of TRLan

[3]Readers are referred to the user guide [Yamazaki et al. 2008] for information on how BLAS (http://www.netlib.org/blas/) and LAPACK (http://www.netlib.org/lapack/) are used in the *a*–TRLan package.

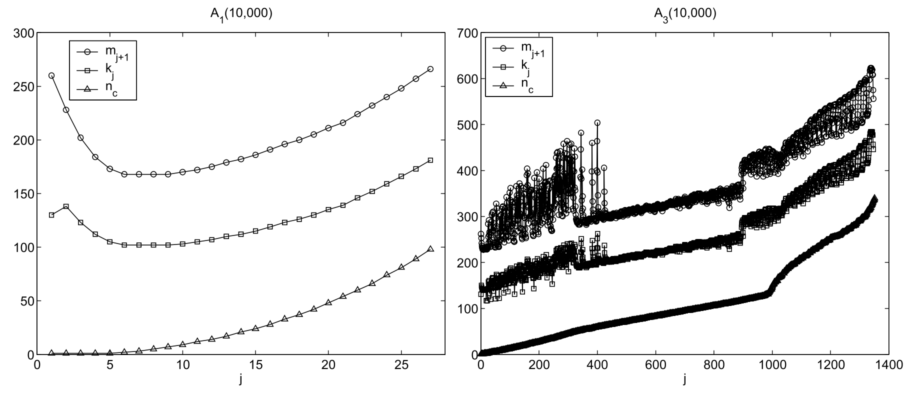

Fig. 6. Projection subspace dimension m_{j+1}, the number k_j of Ritz pairs kept by *a*–TRLan, and the number n_c of converged eigenpairs at the *j*th restart, $m_{\max} = 1000$.

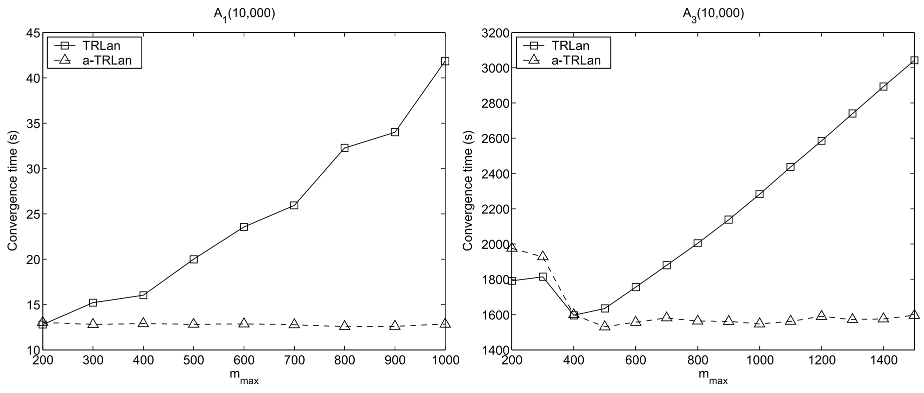

Fig. 7. Elapsed CPU time of TRLan and *a*–TRLan with different $m_{\max}$.

strongly depends on the choice of $m_{\max}$. As a result, *a*–TRLan significantly improves the performance of TRLan, especially when a large $m_{\max}$ is used.

We note that for the test matrix A_1, TRLan achieved its optimal performance using the default subspace dimension $m_{\max} = 2n_d = 200$. However, this default choice may not be always optimal. For the test matrix A_3, optimal performance of TRLan was achieved using $m_{\max} = 400$. It is a nontrivial task to find the optimal $m_{\max}$. On the other hand, when a sufficiently large value of $m_{\max}$ is used, *a*–TRLan automatically determines an appropriate m_{j+1} at every restart. Furthermore, for A_3, *a*–TRLan with $m_{\max} > 400$ improved the optimal performance of TRLan. Similar results were also observed for computing 20 eigenpairs of A_2 (see Figure 1). By setting $m_{\max} = 200$, *a*–TRLan computed the desired eigenpairs in 42.21 seconds in comparison to 46.88 seconds of TRLan using optimal static dimension $m_{\max} = 120$.

Example 2. To demonstrate the effectiveness of *a*–TRLan for solving eigenvalue problems arising from different applications, we tested *a*–TRLan on the matrices available from the University of Florida (UF) sparse matrix

Table I. Description of the Matrices Used in the Numerical Experiments

Name	Description	n	nnz
torsion1	CUTEr optimization problem	40,000	197,608
cfd1	Computational fluid dynamics problem	70,656	1,825,580
Andrews	Computer graphics/vision problem	60,000	760,154
finan512	Multistage stochastic financial modeling	74,752	596,992

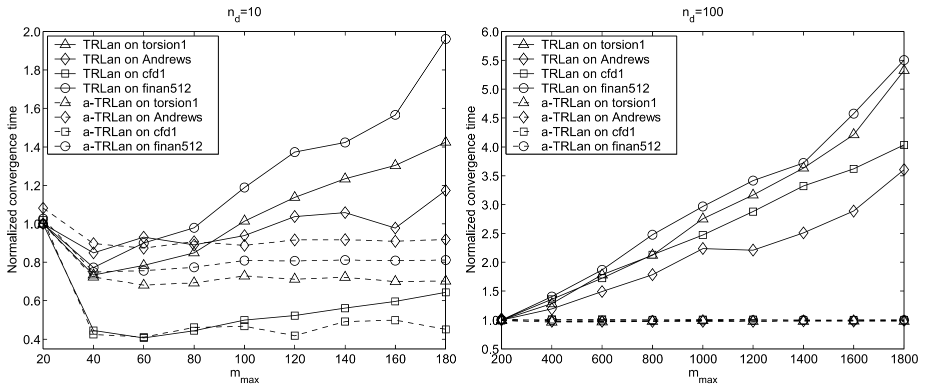

Fig. 8. Performance comparison of TRLan and a–TRLan to compute 10 and 100 eigenpairs of the matrices from the UF sparse matrix collection. For n_d = 10 and 100, the convergence times are normalized by that of TRLan using $m_{\max}$ = 20 and 200, respectively (i.e., 27.9, 14.7, 1169.9, and 48.6 seconds for n_d = 10, and 118.5, 89.0, 2683.1, and 525.9 seconds for n_d = 100).

collection.[4] These matrices were used to evaluate the performance of the Jacobi-Davidson method in PRIMME [Stathopoulos and McCombs 2007]. Some properties of the matrices are shown in Table I. In Figure 8, we show the CPU times required by a–TRLan and TRLan to compute the smallest 10 and 100 eigenvalues of the matrices. These results show that the performance of a–TRLan was essentially independent of $m_{\max}$, and stayed around optimal performance of TRLan as a larger $m_{\max}$ was used.

Example 3. In this example, we study the effect of the relaxation factor ν_j defined as (22). Figure 9 shows the value of ν_j determined by the formula (22) at every restart of a–TRLan to compute the smallest 100 eigenvalues of the matrices A_1 and A_3. The maximum subspace dimension is $m_{\max}$ = 1000. The figure shows that the value of ν_j was adjusted at every restart. Furthermore, for the test matrix A_3, smaller values of ν_j are used to improve the solution convergence. We also observed that the intervals, in which the values of ν_j change abruptly, have some correlation to those in which the number of converged Ritz pairs increases, indicating that the value of ν_j is adjusted when a new Ritz pair converges.

Figure 10 shows the CPU time for computng the smallest 100 eigenvalues of the matrix A_1 using relaxation factor ν_j defined by (22) and static factor ν. The default static value used in TRLan [Wu and Simon 2000b] is ν = 0.4, although the optimal static value is ν = 0.9. The heuristic (22) dynamically adjusts ν_j and

[4]http://www.cise.ufl.edu/research/sparse/matrices/

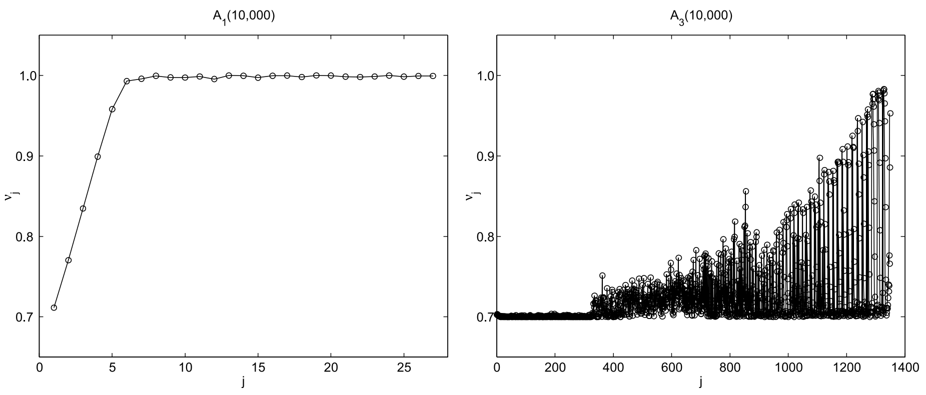

Fig. 9. Relaxation factor ν_j selected by the heuristic (22) at the jth restart.

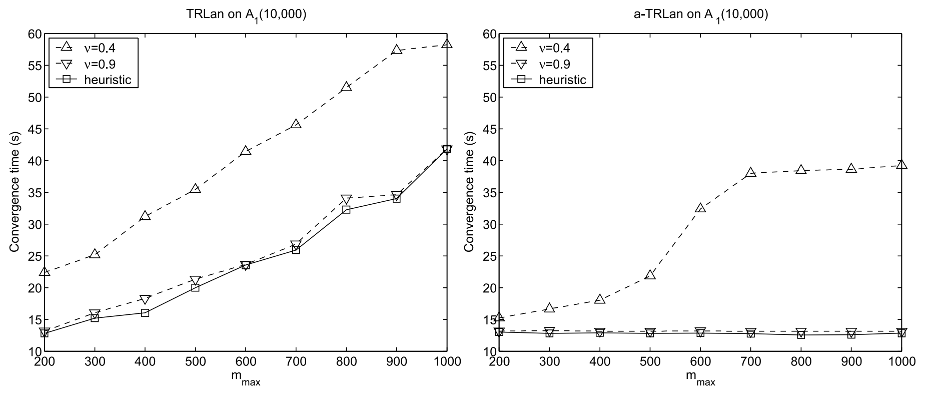

Fig. 10. Elapsed CPU time for computing 100 eigenpairs of A_1 with respect to the difference relaxation factors.

automatically obtains near-optimal performance. We note that for the small static factor of $\nu = 0.4$, a–TRLan keeps large numbers of Ritz pairs at restarts, which lead to unnecessarily large projection subspaces. Similar observations were made for the test matrix A_3 (see Figures 10 and 11).

4.2 Examples from Electronic Structure Calculation

To compute the eigenstates of a Hamiltonian operator, PESCAN uses a PCG-based eigensolver, where the preconditioner is a diagonal matrix constructed based on the kinetic energy portion of the operator [Wang and Zunger 1996]. The PCG-based eigensolver is considered as the state-of-the-art method for this application. Our objective is to compare the CPU times required by PCG-based eigensolver to those required by a–TRLan (preconditioner is not used in a–TRLan). This part of numerical experiments were performed on 16 processors of an IBM POWER 5 system at the National Energy Research Scientific Computing (NERSC) Center. The codes were compiled using the `xlc` compiler and the optimization flag `-O3`.

We considered two quantum dot systems; one consisting of 232 Cadmium atoms and 235 Selenium atoms ($Cd_{232}Se_{235}$), and the other consisting of 534

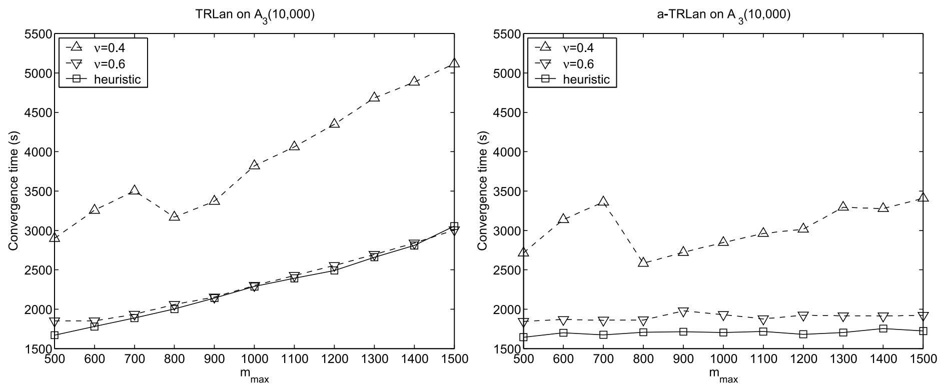

Fig. 11. Elapsed CPU time for computing 100 eigenpairs of A_3 with respect to the difference relaxation factors.

Table II. CPU Time (in seconds) to Compute Eigenpairs of Quantum Dots Using PCG and a–TRLan

	$Cd_{232}Se_{235}$ (n = 75, 645)				$Cd_{534}Se_{527}$ (n = 141, 625)			
	VBM		CBM		VBM		CBM	
n_d	10	30	10	30	10	30	10	30
PCG	39.24	177.21	82.85	210.10	62.63	376.02	193.66	686.25
a–TRLan	39.54	96.49	102.58	125.41	78.63	198.98	233.48	297.20
Speedup	0.99	1.87	0.81	1.68	0.80	1.89	0.83	2.31

Cadmium atoms and 527 Selenium atoms ($Cd_{534}Se_{527}$). PESCAN uses 75,645 and 141,625 planwave bases, which results in Hermitian matrices H of dimensions n = 75,645 and 141,625, respectively. The eigenvalues of H fall into two distinct groups separated by a large band gap between them; the group of smaller eigenvalues is known as the valence band, while the group of larger eigenvalues is called the conduction band. The largest eigenvalues in the valence band are referred to as the Valence Band Maximum (VBM), while the smallest eigenvalues in the conduction band are known as the Conduction Band Minimum (CBM). To evaluate the electrical and optical properties of quantum dot systems, we need to compute the eigenvalues near the band gap [Li and Wang 2004; Schrier and Wang 2006]. We used the folded spectrum method to compute a few eigenpairs in VBM or CBM by computing the smallest eigenvalues of the matrix $A = (H - \lambda_{\text{ref}} I)^2$ with a known reference value λ_{ref}. For the convergence criterion of PCG, we used $\tau = 10^{-5}$, while for that of a–TRLan, τ is adjusted such that at least the same solution accuracy was achieved by the PCG.

Table II shows the required CPU times and speedups gained by a–TRLan to compute different numbers of eigenpairs in VBM and CBM.[5] a–TRLan used the

[5]In Vömel et al. [2008], it is reported that some eigenvalues are missed using ARPACK [Lehoucq et al. 1998]. To avoid missing eigenvalues, five additional eigenpairs are computed with a–TRLan to match the eigenvalues computed with those from PCG in some tests. Furthermore, the required accuracy of the solution computed by a–TRLan is reduced to match the solution accuracy computed by PCG.

heuristic (22) to determine the relaxation factor ν_j, and the maximum subspace dimension was set to be $m_{\max} = 1000$. The numerical results show that to compute a small number of eigenpairs (e.g., $n_d = 10$), PCG was slightly faster than a–TRLan due to the intrincit algorithmic difference of the PCG and Lanczos methods. However, to compute a moderate number of eigenpairs (e.g., $n_d = 30$), a–TRLan performed significantly better than PCG. The performance comparison of these two eigensolvers under limited memory is a subject for future research. For example, PCG used in our experiments requires the memory to store $n_d + 3$ vectors of length n, while a–TRLan requires the memory to store up to $m_{\max}$ vectors. Hence, under limited memory, PCG may be preferred, especially when an effective preconditioner is available.

5. CONCLUSION

The Thick-Restart Lanczos (TRLan) method computes a fixed number of basis vectors before restarting the iteration. Users of TRLan has to carefully select an appropriate basis size. In order to free the users from this difficult task of selecting an appropriate basis size, we proposed an adaptive scheme (a–TRLan) to dynamically determine the projection subspace dimension. The new scheme balances the expected computational cost and solution convergence rate, at every restart. We have developed an open-source software package that implements a–TRLan in C to solve Hermitian eigenvalue problems.

Numerical results have shown that a–TRLan automates the selection of the subspace dimension and improves the performance of TRLan. We have focused on the TRLan method in this article. Other subspace projection eigensolvers such as the implicitly restarted Arnoldi method in ARPACK [Lehoucq et al. 1998] also require users to select an appropriate projection subspace dimension. The idea of the adaptive scheme discussed in this article should also be able to be applied to these eigensolvers.

ACKNOWLEDGMENTS

We gratefully acknowledge fruitful discussions with Sefa Dag, Osni Marques, Matthew Dixon, Christof Vömel, and Nenad Vukmirovic. We thank the anonymous referees for their helpful comments and suggestions to improve the presentation of this article.

REFERENCES

CALVETTI, D., REICHEL, L., AND SORENSEN, D. 1994. An implicitly restarted Lanczos method for large symmetric eigenvalue problems. *Electro. Trans. Numer. Anal. 2*, 1–21.

CANNING, A., WANG, L. W., WILLIAMSON, A., AND ZUNGER, A. 2000. Parallel empirical pseudopotential electronic structure calculations for million atom systems. *J. Comput. Phys. 160*, 1, 29–41.

CROUZEIX, M., PHILIPPE, B., AND SADKANE, M. 1994. The Davidson method. *SIAM J. Sci. Comput. 15*, 62–76.

DEMMEL, J. W. 1997. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, PA.

LANCZOS, C. 1950. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Bur. Stand. 45*, 255–281.

LEHOUCQ, R., SORENSEN, D., AND YANG, C. 1998. *ARPACK Users Guide: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, Philadelphia, PA. Software http://www.caam.rice.edu/software/ARPACK/.

LI, J. AND WANG, L. 2004. First principle study of core/shell structure quantum dots. *Appl. Phys. Lett. 84*, 18, 2648–3650.

MORGAN, P. B. 1996. On restarting the Arnoldi method for large nonsymmetric eigenvalule problems. *Math. Comput. 65*, 215, 1213–1230.

PARLETT, B. N. 1998. *The Symmetric Eigenvalue Problem.* Classics in Applied Mathematics. SIAM, Philadelphia, PA.

PAYNE, M. C., TETER, M. P., ALLAN, D. C., ARIAS, T. A., AND JOANNOPOULOS, J. D. 1992. Iterative minimization techniques for ab-initio total energy calculations: Molecular dynamics and conjugate gradients. *Rev. Mod. Phys. 64*, 1045–1097.

SAAD, Y. 1993. *Numerical Methods for Large Eigenvalue Problems*. Manchester University Press, Manchester, UK.

SCHRIER, J. AND WANG, L. 2006. A systematic first principles study of nanocrystal quantum-dot quantum wells. *Phys. Rev. B 73*, 245332–7.

SORENSEN, D. 1992. Implicit application of polynomial filters in a k-step Arnoldi method. *SIAM J. Mat. Anal. Appl. 13*, 1, 357–385.

STATHOPOULOS, A. AND MCCOMBS, J. R. 2007. Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. Part II: Seeking many eigenvalues. *SIAM J. Sci. Comput. 29*, 5, 2162–2188.

STATHOPOULOS, A., SAAD, Y., AND WU, K. 1998. Dynamic thick restarting of the Davidson and the implicitly restarted Arnoldi methods. *SIAM J. Sci. Comput. 19*, 227–245.

VÖMEL, C., TOMOV, S. Z., MARQUES, O. A., CANNING, A., WANG, L., AND DONGARRA, J. J. 2008. State-of-the-Art eigensolvers for electronic structure calculation of large scale nano-systems. *J. Comput. Phys. 227*, 15, 7113–7124.

WANG, L., CANNING, A., MARQUES, O., AND VOEMEL, C. 2008. The parallel energy scan (PESCAN) code. http://icl.cs.utk.edu/doe-nano/software/pescan/escan.

WANG, L. AND ZUNGER, A. 1994. Solving Schrodinger's equation around a desired energy: Application to silicon quantum dots. *J. Chem. Phys. 100*, 2394–2397.

WANG, L. AND ZUNGER, A. 1996. Pseudopotential theory of nanometer silicon quantum dots application to silicon quantum dots. In *Semiconductor Nanocluster*, P. Kamat and D. Meisel Eds., ACM, New York, NY, 161–207.

WU, K. AND SIMON, H. 2000a. Thick-Restart Lanczos method for large symmetric eigenvalue problems. *SIAM J. Mat. Anal. Appl. 22*, 602–616.

WU, K. AND SIMON, H. 2000b. TRLan user guide. Tech. rep. LBNL-43178, Lawrence Berkeley National Laboratory. Software http://crd.lbl.gov/˜kewu/trlan.html.

YAMAZAKI, I., WU, K., AND SIMON, H. 2008. *a*–TRLan user guide. Tech. rep. LBNL-1288E, Lawrence Berkeley National Laboratory. Software https://codeforge.lbl.gov/projects/trlan/.

Received September 2008; revised October 2009; accepted February 2010