Adaptive Projection Subspace Dimension for the Thick-Restart Lanczos Method[*]

I. Yamazaki[†‡], Z. Bai[‡], H. Simon[†], L.-W. Wang[†], and K. Wu[†]

October 3, 2008

Abstract

The Thick-Restart Lanczos (TRLan) method is an effective method for solving large-scale Hermitian eigenvalue problems. However, its performance strongly depends on the dimension of the projection subspace. In this paper, we propose an objective function to quantify the effectiveness of a chosen subspace dimension, and then introduce an adaptive scheme to dynamically adjust the dimension at each restart. An open-source software package, ν–TRLan, which implements the TRLan method with this adaptive projection subspace dimension is available in the public domain. The numerical results of synthetic eigenvalue problems are presented to demonstrate that ν–TRLan achieves speedups of between 0.9 and 5.1 over the static method using a default subspace dimension. To demonstrate the effectiveness of ν–TRLan in a real application, we apply it to the electronic structure calculations of quantum dots. We show that ν–TRLan can achieve speedups of greater than 1.69 over the state-of-the-art eigensolver for this application, which is based on the Conjugate Gradient method with a powerful preconditioner.

Keywords: Adaptive subspace dimension; Eigenvalue; Lanczos; Thick-restart; Electronic structure calculation.

1 Introduction

The Lanczos method [7] 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 restarted after a fixed number of basis vectors are computed. The performance of such a restarted Lanczos method [8, 24] strongly depends on the user-specified basis size. If the basis size is too small, the method suffers from slow convergence. If it is too large, the computational and memory costs

[*] This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor The Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or The Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or The Regents of the University of California.

[†] Lawrence Berkeley National Laboratory, One Cyclotron road, Berkeley, California 94720.

[‡] University of California, Davis, One Shields avenue, Davis, California 95616.

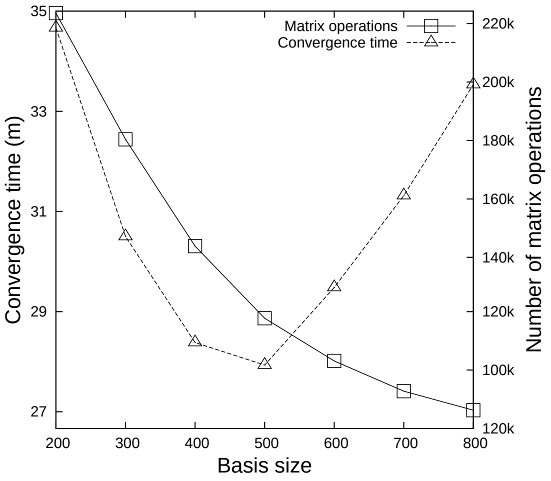

Figure 1: Performance of TRLan with different basis sizes

become expensive. To achieve an optimal performance, it is necessary to select a proper basis size that balances the costs and convergence rate. To demonstrate this delicate task of selecting an appropriate basis size, let us examine the performance of the Thick-Restart Lanczos (TRLan) method [22, 24]. Figure 1 shows the numbers of matrix operations and CPU times in minutes, which are required by TRLan with different basis sizes to compute the smallest 100 eigenvalues of the $10,000 \times 10,000$ diagonal matrix $A = \mathrm{diag}(1, 2^3, 3^3, \ldots, 10000^3)$.[1] As we can see, a larger basis size improves the convergence rate as indicated by a smaller number of matrix operations. However, when the basis size is too large, it becomes expensive to compute the large projection subspace, and the total CPU time starts to increase.

To free users from this difficult task of selecting an appropriate basis size, we propose an adaptive scheme to adjust the dimension of the projection subspace such that optimal performance of the restarted Lanczos method is automatically obtained. We first distinguish between the user-specified maximum basis size and the dimension of the projection subspace used at each restart. We then introduce an objective function that quantifies the effectiveness of a subspace dimension to balance the cost and convergence rate for solving the eigenvalue problem at hand. The subspace dimension is then dynamically adjusted to optimize the objective function.

To demonstrate the effectiveness of this adaptive scheme, we define and implement an objective function for TRLan in this paper. TRLan with this adaptive scheme is referred to as ν–TRLan. The original TRLan software package [23, 24] was implemented in Fortran 90 to solve symmetric eigenvalue problems using a static projection dimension. We have developed a software package in C, which implements both TRLan and ν–TRLan, and have extended it to solve Hermitian eigenvalue problems. Similarly to the original implementation, the message passing interface (MPI) is used to solve eigenvalue problems on distributed memory systems. We present numerical results of synthetic eigenvalue problems to demonstrate that ν–TRLan not only automates the selection of the subspace dimension, but also improves the performance of TRLan that uses an optimal fixed subspace dimension. The open source ν–TRLan software package is publicly available [25].

To show the effectiveness of ν-TRLan in a real application, we integrate it into the Parallel Energy Scan (PESCAN) code, which is used to calculate the electronic structures for semiconductor quantum dots [2, 20] and other applications [9, 14]. The state-of-the-art eigensolver for PESCAN

[1] Additional descriptions of the experiments will be given in Section 4.

is based on the Conjugate Gradient method with a powerful preconditioner [6, 12]. Numerical results show that ν–TRLan can perform significantly better than this state-of-the-art solver with speedups of greater than 1.69.

The rest of this paper is organized as follows: First, we review TRLan in Section 2 and introduce ν–TRLan in Section 3. Then, in Section 4, we present numerical results of synthetic test problems and PESCAN to demonstrate the effectiveness of ν–TRLan. Finally, in Section 5, we conclude with final remarks.

2 Thick-Restart Lanczos method

The Lanczos method [7] is effective for computing a few exterior eigenvalues λ and their corresponding eigenvectors v of a Hermitian matrix A:

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

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

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

These basis vectors satisfy the relation

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

where $Q_i = [q_1, q_2, \ldots, q_i]$, $\beta_i = q_{i+1}^T Aq_i$, e_i is the ith column of the i-dimensional identity matrix, and $T_i = Q_i^T AQ_i$ is an $i \times i$ Rayleigh-Ritz projection of A onto $\mathcal{K}_i(A, q)$. Then, an approximate eigenpair (θ, x) of A is computed from an eigenpair (θ, y) of T_i, where

$$x = Q_i y. \tag{3}$$

The approximate eigenvalue θ and eigenvector x are referred to as a Ritz value and a Ritz vector, respectively. The accuracy of this Ritz pair (θ, x) to approximate an eigenpair (λ, v) of A is measured by its residual norm

$$\|r\|_2 = \|Ax - \theta x\|_2 = \|(AQ_i - Q_iT_i)y\|_2 = \beta_i \|q_{i+1} e_i^T y\|_2 = \beta_i |y(i)|. \tag{4}$$

When the residual norm (4) is less than a prescribed threshold, the Ritz pair (θ, x) is said to be converged. It is well known that Ritz values converge to exterior eigenvalues of A with a subspace dimension $i+1$ that is much smaller than the dimension n of A [11, 13].

The key feature that distinguishes the Lanczos method from other subspace methods is that T_i of (2) is symmetric tridiagonal and of the form

$$T_i = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{i-2} & \alpha_{i-1} & \beta_{i-1} \\ & & & \beta_{i-1} & \alpha_i \end{pmatrix}. \tag{5}$$

Hence, by comparing the ith columns on both sides of (2), we obtain

$$\beta_i q_{i+1} = Aq_i - \alpha_i q_i - \beta_{i-1} q_{i-1}. \tag{6}$$

As a result, in exact arithmetic, the new basis vector q_{i+1} can be computed from two preceding basis vectors, q_{i-1} and q_i. In other words, Aq_i does not have to be orthogonalized against the basis vectors $q_1, q_2, \ldots, q_{i-2}$. This feature is commonly known as the *three-term recurrence*.

However, in practice, when the new basis vector q_{i+1} is computed by (6) on a finite precision machine, orthogonality among the basis vectors is lost even after a small number of iterations. To explicitly maintain orthogonality among the basis vectors, 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 variation of the Gram-Schmidt procedure [13], and it becomes computationally expensive as the subspace dimension $i + 1$ grows. Furthermore, to carry out reorthogonalization and to compute the Ritz vectors x of (3), all the basis vectors Q_i need to be explicitly stored in memory.

To reduce the costs of computing a large subspace, the iteration is restarted after a fixed number of 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 which Ritz values are kept at both ends of spectrum, as shown in Figure 2, where m denotes the basis size at restart.

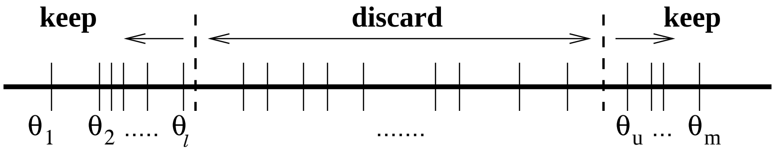

Figure 2: Ritz values to be kept at restart.

The corresponding Ritz vectors to be kept are denoted by

$$\widehat{Q}_k = [\widehat{q}_1, \widehat{q}_2, \ldots, \widehat{q}_k] = Q_m Y_k, \tag{7}$$

where

$$k = \ell + m - u + 1, \tag{8}$$

$$Y_k = [y_1, y_2, \ldots, y_\ell,\ y_u, y_{u+1}, \ldots, y_m], \tag{9}$$

and y_i is the eigenvector of T_{m+1} corresponding to θ_i. Then, TRLan sets these Ritz vectors $\widehat{Q}_k$ as the first k basis vectors for the proceeding iterations after the restart.[2] Furthermore, the Ritz vector q_{m+1} is set to be the $(k+1)$th basis vector $\widehat{q}_{k+1}$. To compute the $(k+2)$th basis vector $\widehat{q}_{k+2}$, TRLan computes $A\widehat{q}_{k+1}$ and explicitly orthonormalizes it against all the previous $k+1$ basis vectors, namely,

$$\widehat{\beta}_{k+1}\widehat{q}_{k+2} = A\widehat{q}_{k+1} - \widehat{Q}_k(\widehat{Q}_k^T A\widehat{q}_{k+1}) - \widehat{q}_{k+1}(\widehat{q}_{k+1}^T A\widehat{q}_{k+1}). \tag{10}$$

Note that $A\widehat{Q}_k$ in the second term on the right-hand side of (10) satisfies the relation:

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

where D_k is the $k \times k$ diagonal matrix whose diagonal elements are the kept Ritz values, and $s = Y_k^T e_m$. Thus, the coefficients $\widehat{Q}_k^T A\widehat{q}_{k+1}$ in (10) can be computed efficiently:

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

```
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.  $\alpha_{k+1} = q_{k+1}^T p$
        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 $i = k+2, k+3, \ldots, m$
        b.      $p = Aq_i$
        c.      $\alpha_i = q_i^T p$
        d.      $p = p - \alpha_i q_i - \beta_{i-1} q_{i-1}$
        e.      reorthogonalize $p$ if necessary.[3]
        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 (4).
        b.  if stopping criteria is satisfied then
        c.      compute desired Ritz vectors and exit.
        d.  else restart:
        e.      select $(\ell_{j+1}, u_{j+1}, m_{j+1})$ and compute $k_{j+1}$ of (8).
        f.      set $k = k_{j+1}$ and $m = m_{j+1}$
        g.      compute eigenvectors $Y_k$ of (9).
        h.      compute Ritz vectors $\widehat{Q}_k$ of (7).
        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 $\pi_1, \pi_2, \ldots, \pi_k = 1, 2, \ldots, \ell, u, u+1, \ldots, m$.
        k.  end if
end for
```

Figure 3: Pseudocode of TRLan algorithm.

This is the only iteration of TRLan that does not follow the three-term recurrence (6). Hence, at the ith iteration after the restart, the new basis vector $\widehat{q}_{k+i+1}$ satisfies the relation:

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

[2] To distinguish the basis vectors computed after the restart from those computed before the restart, we put a hat over the basis vectors computed after the restart.

[3] Perturbation of p may be needed when p is in the invariant space of A, and $p \approx 0$.

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

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

Figure 3 shows the pseudocode of the TRLan algorithm with the initial projection subspace dimension m_1. A more detailed description of TRLan can be found in [22].

Note that some approaches were proposed to relax the requirement for orthogonality among the basis vectors Q_i, such as the Lanczos method without reorthogonalization [4], the Lanczos method with partial reorthogonalization [15, 19], and the Conjugate Gradient (CG) method [12]. However, they all have their own difficulties. Without reorthogonalization, the Lanczos method computes spurious eigenvalues and requires a careful examination to identify them in the final solution. With partial reorthogonalization, the Lanczos method can produce inaccurate eigenvectors. The CG method for eigenvalue problems typically requires a good preconditioner, which may not be easily available. For these reasons, methods that explicitly maintain orthogonality among the basis vectors are more commonly used.

Besides TRLan, 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 lot of useful information at restart and results in slow convergence. To preserve more information, other schemes have been proposed that keep more than one vector at restart. For example, the implicitly restart Lanczos method [1] keeps a fixed number of vectors at restart that approximately span a space containing the desired Ritz vectors by filtering out the unwanted ones. TRLan similarly keeps multiple vectors at restart, but allows more explicit control over which Ritz vectors are kept.

3 Adaptive scheme for adjusting 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 indicate the kept Ritz vectors and dimension of the next projection subspace. In the original implementation of TRLan [23, 24], the kept Ritz vectors are selected to maximize the rate of the solution convergence over the next restart-loop, while the projection subspace dimension m_{j+1} is fixed to be the user-specified maximum basis size m_{max} for all j. As discussed in Section 1, this requires users to carefully select the basis size in order to achieve a good performance of TRLan for each eigenvalue problem at hand. To free users from this difficult task of selecting an appropriate basis size, in this section, we propose an adaptive scheme for adjusting the dimension of the projection subspace at each restart. This is based on the definition of an *objective function* of a triplet (ℓ, u, m):

$$f(\ell, u, m),$$

which quantifies the effectiveness of the triplet to achieve optimal performance. This objective function is defined based on the careful examination of the computational cost (Sections 3.1) and solution convergence rate (Section 3.2) over the $(j+1)$-th restart-loop. In Section 3.3, we propose a specific objective function that measures the effectiveness of the triplet to balance the costs and convergence rate. We also discuss practical issues to implement this adaptive scheme in Sections 3.4 and 3.5.

3.1 Computational cost

The dominant computational costs of ν–TRLan are:

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) [6]. For simplicity, we consider the full reorthogonalization.[4] The aggregated cost of all reorthogonalizations is approximately given by
$$\sum_{i=k}^{m-1} 4ni = 2n(m-k)(k+m-1) \text{ flops.} \tag{11}$$

2. Ritz vector computation (Step 3.h in Figure 3). The computation of the Ritz vectors
$$\widehat{Q}_k = Q_m Y_k$$
that are kept at the j-th restart requires approximately
$$2nmk \text{ flops,} \tag{12}$$
where Q_m and Y_k are an $n \times m_j$ and an $m_j \times k$ matrix, respectively.

Besides these costs, the computational cost of a TRLan iteration can be dominated by the matrix-vector multiplication. However, the cost of each matrix-vector multiplication is independent of the subspace dimension $m + 1$. As the subspace dimension grows, we expect the computational cost of the reorthogonalization process to dominate that of the restart-loop. Thus, by summing (11) and (12), we use the following formula to measure the total cost for the $(j + 1)$-th restart-loop:[5]

$$2n(m-k)(k+m-1) + 2nmk = 2n((m-k)(m+k-1) + mk). \tag{13}$$

3.2 Convergence factor

To be concrete, our discussion here focuses on computing the smallest n_d eigenvalues. A similar discussion can be made for computing the largest eigenvalues.

Let ρ_j be the reduction factor of the residual norm (4) of the smallest unconvergent Ritz value at the end of the $(j + 1)$-th restart-loop, i.e., $\|r_{j+1}\|_2 = \rho_j \|r_j\|_2$, where $\|r_j\|_2$ is the residual norm at the j-th restart. The factor ρ_j of the Ritz pair (θ_1, x_1) has been extensively studied [6, 11, 13]. A key result of the convergence analysis is that after m Lanczos iterations without restart, an upper-bound of ρ_j is given by $\frac{1}{\mathcal{C}_m(1+2\gamma)}$, where $\mathcal{C}_m$ is the Chebyshev polynomial of degree m, $\gamma = \frac{\lambda_2-\lambda_1}{\lambda_n-\lambda_2}$ is a gap ratio of eigenvalues λ_j of A, and $\lambda_1 \le \lambda_2 \le \ldots \le \lambda_n$ [13]. We focus on the worst case
$$\rho_j = \frac{1}{\mathcal{C}_m(1+2\gamma)}. \tag{14}$$

[4] TRLan and ν–TRLan implement an algorithm similar to the one suggested in [11] to determine when to apply the reorthogonalization. In practice, we have observed that most of the new basis vectors need to be reorthogonalized.

[5] ν–TRLan offers an option for users to include the cost of matrix-vector multiplication in (13).

To incorporate the effect of restart on the reduction factor ρ_j of (14), let us assume that Ritz pairs which are kept at the j-th restart have been converged exactly to the corresponding eigenpairs of the matrix A. Then, according to the analysis of Morgan [10], the convergent Ritz vectors deflate the corresponding spectrum of A, and the factor ρ_j of the smallest unkept Ritz value over the $(j+1)$-th restart-loop is given by

$$\rho_j = \frac{1}{\mathcal{C}_{m-k}(1+2\gamma)}, \tag{15}$$

where the gap ratio γ is defined as

$$\gamma = \frac{\lambda_{\ell+2} - \lambda_{\ell+1}}{\lambda_{n-m+u-1} - \lambda_{\ell+1}}. \tag{16}$$

We compute an approximation of the reduction factor ρ_j as follows:

$$\rho_j = \frac{1}{C_{m-k}(1+2\gamma)} \approx \frac{1}{\cosh(2(m-k)\sqrt{\gamma})}, \tag{17}$$

where following [5, Lemma 5.7], the Chebyshev polynomial $\mathcal{C}_{m-k}(1+2\gamma)$ is approximated by

$$\mathcal{C}_{m-k}(1+2\gamma) = \cosh((m-k)\text{arccosh}(1+2\gamma)) \approx \cosh(2(m-k)\sqrt{\gamma}).$$

We define an additional factor

$$\omega_j = \frac{1}{\rho_j}. \tag{18}$$

Thus, the residue norm is expected to be reduced by the factor ω_j over the $(j+1)$-th restart-loop.

3.3 Objective function

We now define an objective function for a triplet (ℓ, u, m) as the ratio of the reduction factor (18) over the computational cost (13):

$$f(\ell, u, m) = \frac{\cosh(2(m-k)\sqrt{\gamma})}{2n((m-k)(m+k-1)+mk)}. \tag{19}$$

An optimal triplet $(\ell_{opt}, u_{opt}, m_{opt})$ maximizes the value of the objective function (19),

$$(\ell_{opt}, u_{opt}, m_{opt}) = \arg\max f(\ell, u, m), \tag{20}$$

subject to

$$0 \le \ell \le a_j, \quad m_j - b_j + 1 \le u \le m_j + 1, \quad k \le m \le m_{max}, \tag{21}$$

where a_j and b_j are the numbers of the smallest and largest convergent Ritz values, respectively; k is the number of the kept Ritz pairs and given by (8); and m_{max} is a prescribed maximum basis size. When $\ell = 0$ or $u = m_j + 1$, the smallest or largest Ritz values, respectively, are not kept. The solution to the optimization problem (20) maximizes the expected reduction of the residue norm per flop over the next restart-loop. Hence, it is expected that $(\ell_{opt}, u_{opt}, m_{opt})$ balances the cost and solution convergence, and achieves optimal performance for the next restart-loop. Therefore, for Step 3.e of the pseudocode in Figure 3, ν–TRLan sets $(\ell_{j+1}, u_{j+1}, m_{j+1})$ to be $(\ell_{opt}, u_{opt}, m_{opt})$.

The optimization problem (20) is solved by explicitly computing $f(\ell, u, m)$ for all possible combinations of triplets (ℓ, u, m). Even though the cost of this approach is $O(m_{max}^3)$, it is expected to be insignificant to the total computational cost since m_{max} is typically much smaller than n.

3.4 Practical objective function

In practice, the exact eigenvalues of A are not available. Hence, we replace the eigenvalues λ_j in (16) with the computed Ritz values θ_j and define an *effective* gap ratio γ_e as follows:

$$\gamma_e = \frac{\theta_{\ell+1} - \theta_{t_j}}{\theta_{u-1} - \theta_{\ell+1}}, \tag{22}$$

where t_j specifies the target Ritz value at the j-th restart, i.e., $t_j = c_j + 1 \leq \ell$ with c_j being the number of the smallest Ritz values that have converged to satisfy a required solution accuracy.

Note that the convergence rate of the target Ritz pair can be improved by keeping the Ritz pairs around the target even though they have not yet been 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 (21), we enforce the following constraints on the indices ℓ and u for the definition of γ_e in (22):

$$c_j \leq \ell \quad \text{and} \quad u \leq m_j + 1. \tag{23}$$

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:

$$g_j = \nu \cdot (m_j - c_j), \tag{24}$$

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

Combining the constraints (23) and (24), we arrive at the following ranges of the indices ℓ and u:

$$c_j \leq \ell \leq m_j + 1 - g_j \quad \text{and} \quad \ell + g_j \leq u \leq m_j + 1. \tag{25}$$

Once both ℓ and u are determined, the valid range of the next projection subspace dimension m is

$$k < m \leq m_{max}, \tag{26}$$

where k is given by (8) and m_{max} is the prescribed maximum basis size.

Since Ritz pairs that have not yet been converged are now used to define the effective gap ratio γ_e, the factor (18) does not accurately measure the reduction of the target residual norm. In particular, the cosh function in (18) grows quickly with respect to γ_e, and the error in γ_e to approximate γ can be greatly amplified. Alternatively, it was found in [22] that the following formula accurately measures the progress of the target Ritz pair after the $(m-k)$ iterations:

$$\omega_j \approx 2(m-k)\sqrt{\gamma_e}. \tag{27}$$

Note that for a typical eigenvalue problem with $0 < \gamma \ll 1$, the Chebyshev polynomial $\mathcal{C}_{m-k}(1 + 2\gamma_e)$ can be approximated by $\frac{1}{2}(1 + 2(m-k)\sqrt{\gamma_e})$ [5, Lemma 5.7].

Thus, for a practical implementation of (19), we use the following objective function:

$$f(\ell, u, m) = \frac{(m-k)\sqrt{\gamma_e}}{n((m-k)(m+k-1) + mk)}, \tag{28}$$

where the triplet (ℓ, u, m) is subject to the constraints (25) and (26), and the effective gap ration γ_e is given by (22).

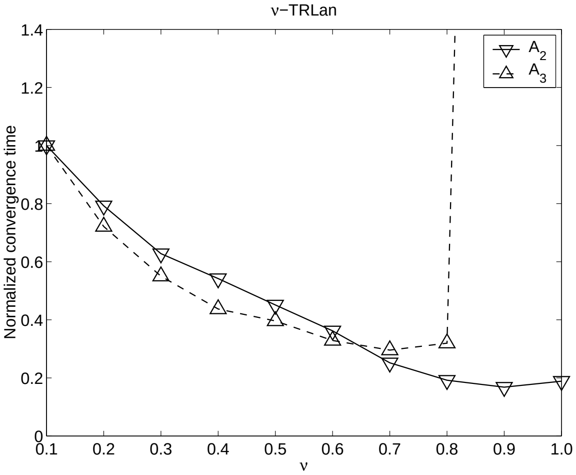

Figure 4: Effect of relaxation factor ν on performance of ν–TRLan.

The original implementation of TRLan [24, 23] selects the indices ℓ_{j+1} and u_{j+1} that maximize the reduction factor (27), while the dimension of the projection subspace is fixed, i.e., $m_{j+1} = m_{max}$ for all j. Hence, the objective function is given by

$$g(\ell, u) = (m_{max} - k)\sqrt{\gamma_e}, \tag{29}$$

where (ℓ, u) is subject to the constraint (25). A similar restart scheme is used for the thick-restarted Davidson method [18].

We remark that an adaptive scheme to adjust the dimension of the projection subspace of the Davidson method was previously studied [3]. In their scheme, 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., $\|r_j\|_2/\|r_{j-1}\|_2$, where $\|r_j\|_2$ is the residual norm at the end of the jth iteration) grows significantly. Our adaptive scheme, on the other hand, attempts to optimize the performance over the proceeding restart-loop based on careful examination of the solution convergence rate.

3.5 Heuristic for the relaxation factor ν

The effectiveness of both objective functions (28) and (29) depends on the relaxation factor ν of (24). To demonstrate the effects of ν, Figure 4 shows the CPU time for ν–TRLan using $m_{max} = 500$ and different ν to compute the smallest $n_d = 100$ eigenvalues of two $10,000 \times 10,000$ diagonal matrices, $A_2 = \text{diag}(1, 2^2, 3^2, \ldots, 10000^2)$ and $A_3 = \text{diag}(1, 2^3, 3^3, \ldots, 10000^3)$. The CPU times are normalized by that of $\nu = 0.1$. The figure clearly indicates the strong impact of ν on the performance of ν–TRLan. It also shows that optimal performance of ν–TRLan is achieved with different ν for A_2 and A_3; i.e., optimal performance is achieved with $\nu = 0.9$ for A_2 and with $\nu = 0.7$ for A_3. In the original TRLan implementation [23, 24], the relaxation factor is fixed at $\nu = 0.4$.

To eliminate the need to search for an optimal ν for each problem at hand, we design a heuristic to dynamically adjust ν based on the observed solution convergence rate. Specifically, we select the relaxation factor ν_j at the j-th restart by considering the factor $\frac{\|r_j\|_2}{\|r_{j-1}\|_2}$ of the $(j-1)$-th target Ritz pair $(\theta_{t_{j-1}}, x_{t_{j-1}})$, where $\|r_j\|_2$ is the residual norm of the target at the j-th restart. Then, we define an *observed* gap ratio γ_o over the j-th restart-loop as

$$\gamma_o = \left(\frac{\operatorname{arccosh} \frac{\|r_{j-1}\|_2}{\|r_j\|_2}}{2(m_j - k_j)} \right)^2 . \tag{30}$$

Recall that the factor ρ_j of the target Ritz pair was previously defined by (17) using the gap ratio γ. At the same time, we found that a *desired* gap ratio γ_d which achieves good performance of ν–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 for the relative residual norm of the convergent Ritz pairs, $\|A\|_2$ is approximated by the largest absolute value of all the convergent Ritz values, and $\bar{m}$ is the average projection subspace dimension of the previous restart-loops. Thus, γ_d is computed as

$$\gamma_d = \left(\frac{\operatorname{arccosh} \frac{\|r_{j-1}\|_2}{\tau \|A\|_2}}{4\bar{m}} \right)^2 . \tag{31}$$

When the observed gap ratio γ_o is smaller than the desired gap ratio γ_d, it indicates a 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 according to the observed gap ratio, we introduce the following heuristic:

$$\nu_j = \nu_\ell + (1 - \nu_\ell) \left(\frac{2}{\pi} \right) \arctan \frac{\gamma_o}{\gamma_d}, \tag{32}$$

where ν_ℓ is a lower-bound on ν_j, $0 \le \nu_\ell \le 1$. In Figure 4, ν–TRLan was most effective when ν was between 0.7 and 1.0. Hence, we set the default lower-bound to be $\nu_\ell = 0.7$.

We note that when the target 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 (30) is not defined. In this case, we set to the default value $\nu_j = 0.7$. We present the numerical results of the heuristic (32) in Section 4.

4 Numerical experiments

In this section, we present numerical results to demonstrate the effectiveness of ν–TRLan. First, to compare the performance of ν–TRLan and TRLan, we consider two synthetic diagonal matrices,

1. $A_1(n) = \mathrm{diag}(1, 2, \ldots, n)$, and
2. $A_2(n) = \mathrm{diag}(1^2, 2^2, \ldots, n^2)$.

These matrices provide good test problems due to the following properties:

1. The matrices have known eigenvalues and eigenvectors.
2. The condition numbers of the matrices can be easily controlled by adjusting the matrix sizes n.

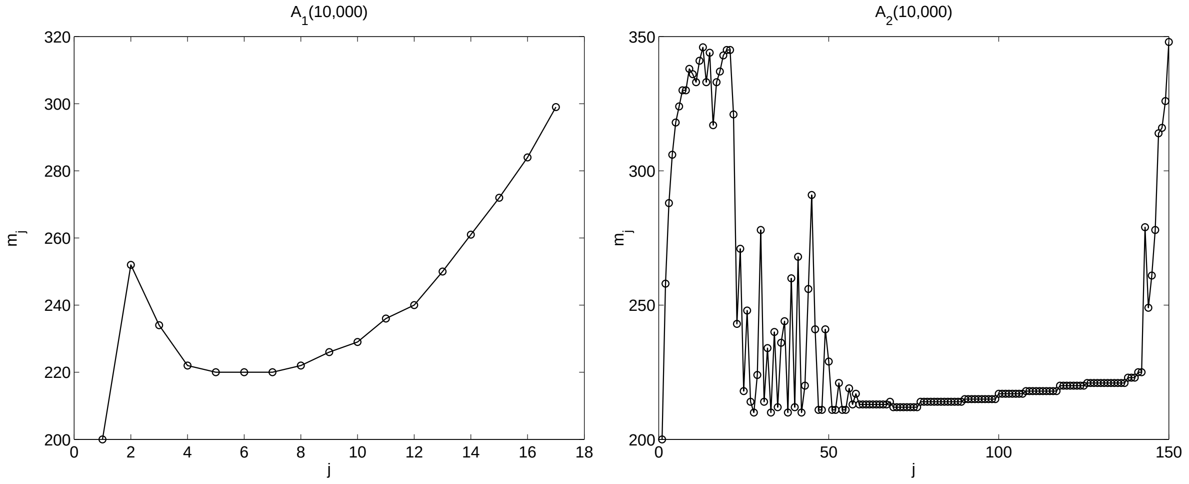

Figure 5: Projection subspace dimension m_j selected by ν–TRLan at the j-th restart.

3. The matrices are accessed only through the matrix-vector multiplications. Thus, aside from the cost of the matrix-vector multiplication, the performance of TRLan and ν–TRLan to compute eigenpairs of the test matrix is identical to that to compute the eigenpairs of any Hermitian matrix having the same eigenvalue distribution.

For our numerical experiments, we use the test matrices $A_1(n)$ and $A_2(n)$ of dimension $n = 10,000$. The initial vector to the Lanczos iteration is a vector with all of its entries set to be one. The iteration is continued until the following relative accuracy is achieved: $\|Ax - \theta x\|_2 \leq \tau \|A\|_2$, where $\tau = \sqrt{\epsilon} = 2^{-26} \approx 1.4901 \times 10^{-8}$, and $\|A\|_2$ is approximated by the largest absolute value of the computed Ritz values. The numerical experiments were conducted on an HP Itanium2 workstation with a 1.5GHz CPU and 2GB of RAM. The codes were complied with the `icc` compiler from Intel Math Kernel Library and with the optimization flag `-O3`.

We first examine the projection subspace dimension m_j that is dynamically selected by ν–TRLan at the j-th restart. Figure 5 shows the selected m_j for computing the smallest $n_d = 100$ eigenvalues of A_1 and A_2. The maximum basis size is set to be $m_{max} = 1,000$. Our implementation initially sets $m_1 = \min(2n_d, m_{max})$, and enforces $\ell_{j+1} \geq n_d$ and $u_{j+1} \leq m_j - 1$. The figure clearly shows that the subspace dimension is adjusted at every restart.

Correspondingly, Table 1 compares the total CPU times required to solve the above eigenvalue problems with TRLan and ν–TRLan. The basis size for TRLan is fixed to be its default value, which is twice the number of desired eigenpairs, i.e., $m_{max} = 2n_d = 200$ [23]. This default basis size coincidently achieves optimal performance of TRLan for these eigenvalue problems. The numerical results demonstrate that ν–TRLan can even improve optimal performance of TRLan.

	A_1	A_2
TRLan	13.75	101.03
ν–TRLan	12.53	103.36
Speedup	1.10	0.98

Table 1: CPU time in second to compute 100 eigenpairs with $m_{max} = 1,000$.

Next, we examine the performance of TRLan and ν–TRLan with respect to different maximum basis sizes m_{max}. Figure 6 compares the total CPU times required by TRLan and ν–TRLan to

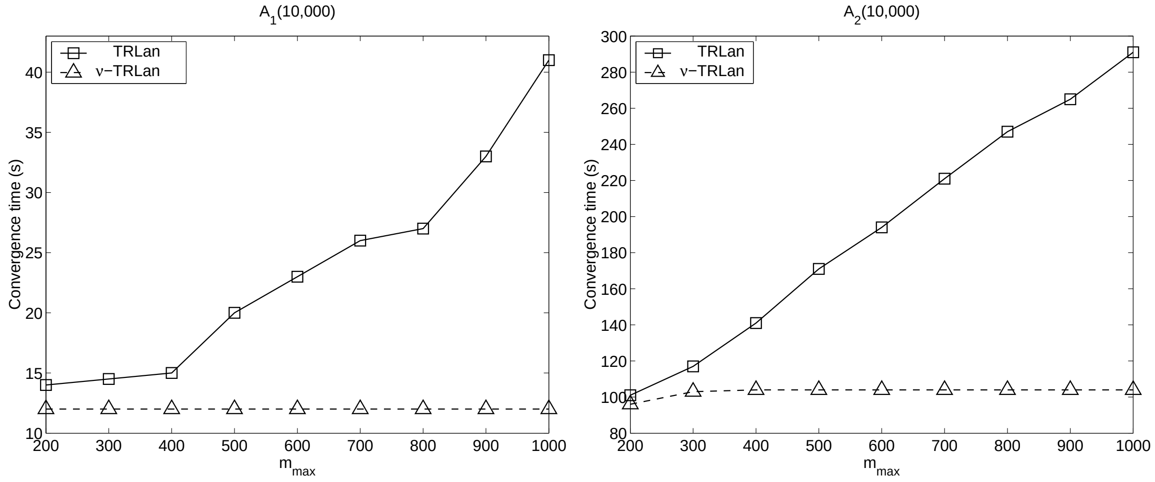

Figure 6: Performance comparison of TRLan and ν–TRLan with different m_{max}.

compute the smallest $n_d = 100$ eigenpairs of the matrices A_1 and A_2 using different m_{max}. TRLan always computes m_{max} basis vectors before restart. The figure shows that the performance of ν–TRLan is largely independent of m_{max}, while the performance of TRLan strongly depends on it. As a result, ν–TRLan significantly improves the performance of TRLan, especially when a large m_{max} is used. The speedups gained by ν–TRLan are up to 3.42.

Next, we examine the CPU times spent in reorthogonalization and restart,[6] which are used for designing the total expected cost (13). Table 2 shows the CPU times required to compute the $n_d = 100$ smallest eigenvalues of A_2. Since the test matrix used here is a diagonal matrix, reorthogonalization and restart indeed dominate the overall CPU time, verifying that (13) is a good measure of the total cost. We note that as long as the cost of the matrix-vector multiplication is proportional to the matrix dimension n, the solution of the optimization problem (28) is not affected even if the cost of the multiplication is included in the total cost (13).

	TRLan			ν–TRLan		
Basis size m_{max}	200	400	1,000	200	400	1,000
Reorth.	57.95	101.15	242.99	55.75	64.18	64.32
Restart	40.13	39.04	44.17	39.18	38.28	38.52
Total	100.03	141.75	291.51	96.72	104.01	104.36

Table 2: CPU time in second to compute 100 eigenpairs of A_2.

Finally, to study the performance of ν–TRLan to compute a smaller number of eigenpairs, we compute the smallest $n_d = 20$ eigenvalues of the test matrix A_2. For this eigenvalue problem, TRLan required about 356 seconds using the default basis size of $m_{max} = 2n_d = 40$, but the solution time can be reduced by using a larger basis size and improving the convergence rate. Optimal performance of both TRLan and ν–TRLan is achieved around $m_{max} = 150$. Table 3 shows the CPU times to solve this eigenvalue problem using the basis size around this optimal basis size. The table also shows the speedup that is gained by ν–TRLan over TRLan for the prescribed m_{max}. We see that the performance of ν–TRLan stays at optimal for $m_{max} \geq 150$. As

[6] The time to restart is dominated by that to compute the Ritz vectors.

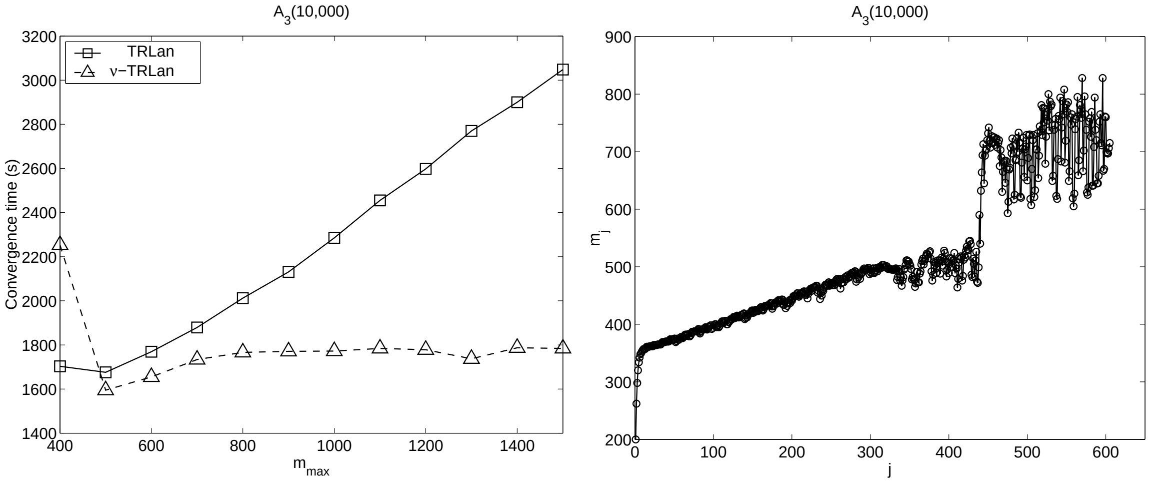

Figure 7: Performance data of ν–TRLan, the convergence time with different m_{max} (Left) and projection subspace dimension m_j with $m_{max} = 1,000$ (Right), to compute 100 eigenpairs of A_3.

a result, the performance advantage of ν–TRLan is maintained for computing the smaller number of eigenpairs (i.e., ν–TRLan with $m_{max} = 500$ achieves the speedup of 5.15 over TRLan with the default $m_{max} = 40$).

Basis size m_{max}	100	150	200	250	500
Reorth.	44.77	38.55	38.84	39.05	38.68
Restart	30.54	26.93	27.12	27.09	27.11
Total	78.86	68.67	69.22	69.27	69.06
Speedup	0.87	0.96	1.05	1.29	2.31

Table 3: CPU time in second to compute 20 eigenpairs of A_2 using ν–TRLan.

Now, to examine the performance of ν–TRLan for solving ill-conditioned eigenvalue problems, we consider one additional synthetic matrix, $A_3(n) = \text{diag}(1^3, 2^3, \ldots, n^3)$. We compute the smallest $n_d = 100$ eigenpairs of $A_3(10,000)$, which provides an extremely ill-conditioned problem. Since $\|A_3(10,000)\|_2 = 10^{12}$, the iteration is continued until the relative accuracy $\tau = 10^{-13}$ of the residual norm is achieved.

For solving this eigenvalue problem with A_3, TRLan required about $2,079$ seconds with the default basis size of $m_{max} = 200$, but the convergence time can be improved by using a larger m_{max}. The left plot of Figure 7 compares the total CPU times required by TRLan and ν–TRLan with different m_{max} around the optimal basis size $m_{max} = 500$. The figure shows that the CPU time required by ν–TRLan increases slightly as the maximum basis size m_{max} is increased from the optimal basis size. However, the performance of ν–TRLan is still less dependent on m_{max} than TRLan. As a result, ν–TRLan considerably improves the performance of TRLan (i.e., ν–TRLan with $m_{max} = 1000$ achieves a speedup of 1.73 over TRLan using the default basis size of $m_{max} = 200$). The right plot of Figure 7 shows how the projection subspace dimension m_j is adjusted by ν–TRLan at the j-th restart, where the maximum basis size is set to be $m_{max} = 1,000$.

We now study the impact of the heuristic (32) to dynamically adjust the relaxation factor ν_j on the performance of TRLan and ν–TRLan. Figure 8 shows the value of ν_j selected by (32) at every restart of ν–TRLan to compute the smallest $n_d = 100$ eigenvalues of the matrices A_2 and A_3.

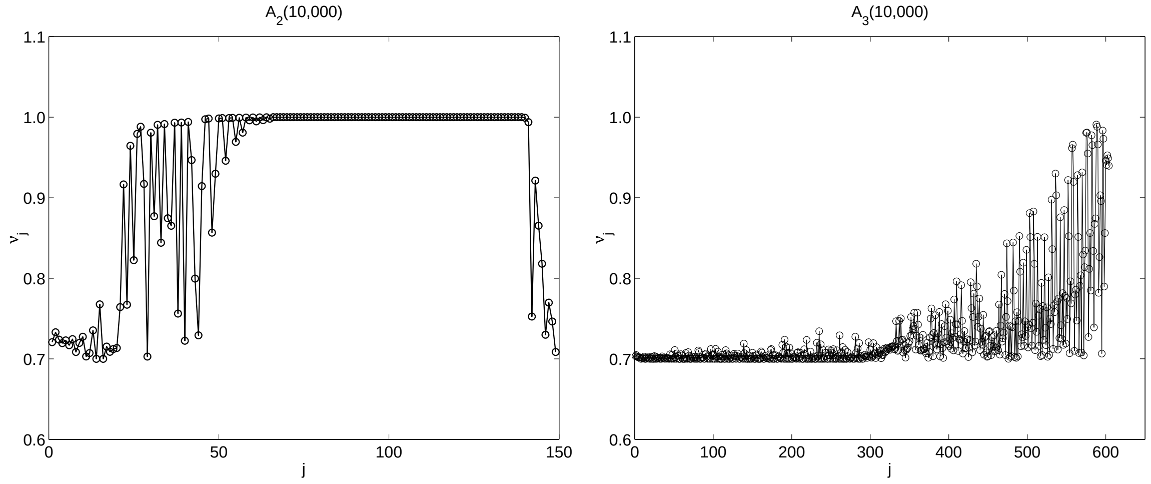

Figure 8: Relaxation factor ν_j selected by the heuristic (32) at the j-th restart.

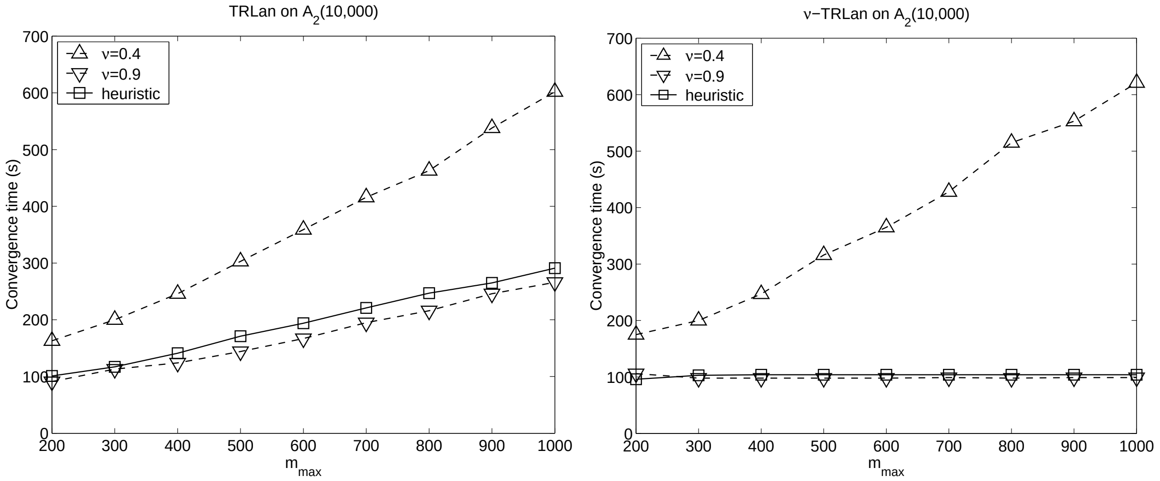

Figure 9: Performance of the heuristic (32) for computing 100 eigenpairs of A_2.

The maximum basis size is set to be $m_{max} = 1,000$. The figure clearly shows that the value of ν_j is adjusted at every restart. Furthermore, smaller values of ν_j are used for ill-conditioned A_3 to improve the solution convergence.

Figure 9 examines the performance of (32) by showing the CPU times required to compute the smallest $n_d = 100$ eigenvalues of the test matrix A_2 using (32) and different maximum basis size m_{max}. It also compares the CPU times with those required when a static relaxation factor is used, $\nu_j = \nu$ for all j. We show the performance with $\nu = 0.4$, which is the default value used in the original implementation of TRLan [23, 24], and with $\nu = 0.9$, which is the optimal for this problem. The figure clearly indicates that the performance of both TRLan and TRLan$_\nu$ strongly depends on the static parameter ν. The CPU time with the heuristic (32) is increased from that with the optimal $\nu = 0.9$ by small factors of less than 1.15 and 1.10 for TRLan and ν–TRLan, respectively. We note that for the small factor of $\nu = 0.4$, ν–TRLan keeps large numbers of unconvergent Ritz pairs at restart, which lead to unnecessary large projection subspaces. Hence, the performance of ν–TRLan is close to that of TRLan.

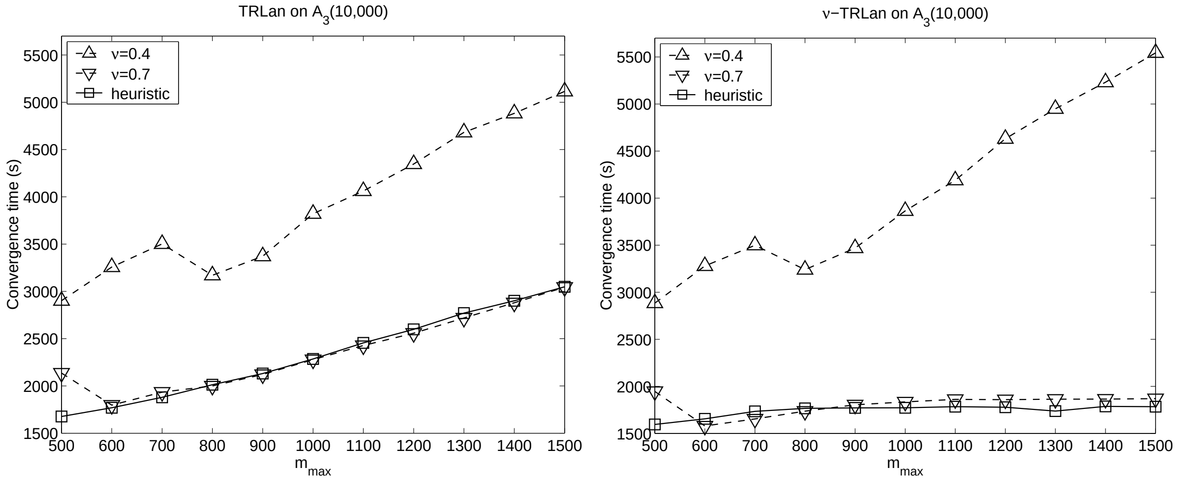

Figure 10: Performance of the heuristic (32) for computing 100 eigenpairs of A_3.

In Figure 10, we show the CPU time required to compute the smallest $n_d = 100$ eigenvalues of the test matrix A_3 using the heuristic (32) and different values of m_{max}, and compare it with the CPU time using a static $\nu_j = 0.4$, and also with $\nu_j = 0.6$. We note that $\nu_j = 0.4$ is the default value used in the original TRLan implementation, and $\nu_j = 0.6$ achieved optimal performance for A_3 in Fig 4. The figure shows that the CPU time with (32) is about the same as that with optimal $\nu_j = 0.6$ for both TRLan and ν–TRLan. These numerical results indicate that the heuristic (32) automatically obtains a near-optimal performance of TRLan and ν–TRLan, and frees users from searching for an appropriate ν for each problem.

As our last example, we demonstrate the effectiveness of ν–TRLan in a real application by presenting numerical results of Parallel Energy Scan (PESCAN) code, which is used to calculate the electronic structures for semiconductor quantum dots [2, 20] and other applications [9, 14]. Specifically, we replace the existing eigensolver in PESCAN, which is based on the Preconditioned Conjugate Gradient (PCG) method [6, 12], with ν–TRLan. PESCAN implements a powerful preconditioner for PCG by solving the kinetic energy portion of the Hamiltonian operator [21]. This preconditioner significantly improves the PCG convergence rate, and PCG is the state-of-the-art method for this application.

The eigenvalues of a Hermitian matrix H from PESCAN 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. Typically, the eigenvalues of interests are those near the band gap, and they are used to evaluate the electrical and optical properties of quantum dot systems [9, 14]. 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 compare the performance of PCG and ν–TRLan, we use the folded spectrum method to compute a few eigenpairs in VBM or CBM by computing the smallest eigenvalues of the matrix $A = (H - \lambda_{ref} I)^2$ with a known reference value λ_{ref}.

Table 4 shows the CPU times required to compute different numbers n_d of eigenpairs in VBM and CBM for the quantum dot system consisting of 534 Cadmium atoms and 527 Selenium atoms.[7] The numerical performance data were collected on an IBM SP RS/6000 system at

[7] In some test cases, additional 5 eigenpairs are computed with ν–TRLan to match the eigenvalues computed with those from PCG.

the National Energy Research Scientific Computing Center (NERSC) using one node with 16 processors. The computation uses $141,625$ planwave bases, which results in a Hermitian matrix H of dimension $n = 141,625$. ν–TRLan uses the heuristic (32) to select ν_j, and the maximum basis size is set to be $m_{max} = 1,000$. In the table, speedup is that gained by ν–TRLan over PCG. We see that ν–TRLan performs significantly better than PCG for computing 30 or more eigenpairs.

	VBM			CBM		
n_d	10	30	100	10	30	100
PCG	987	$3,679$	$9,892$	311	$2,026$	$> 18,000$
ν–TRLan	$1,629$	$3,230$	$7,441$	790	$1,459$	$5,343$
Speedup	0.61	1.17	1.33	0.40	1.39	$--$

Table 4: CPU time in second to compute eigenpairs of $Cd_{534}Se_{527}$ using PCG and ν–TRLan.

In Table 5, we also show the CPU times required to compute 30 eigenvalues in VBM of two other Cadmium Selenium quantum dots. The dimension of the Hermitian matrix is denoted by n. We see that the relative advantage of ν–TRLan is maintained as the matrix size changes.

	n	PCG	ν–TRLan	Speedup
$Cd_{83}Se_{81}$	34,143	459	272	1.69
$Cd_{232}Se_{235}$	75,645	$1,040$	971	1.07

Table 5: CPU time in second to solve eigenvalue problems for PESCAN using PCG and ν–TRLan.

5 Conclusion

The original implementation of the Thick-Restart Lanczos (TRLan) method computes a fixed number of basis vectors before restarting the iteration. This requires users to carefully select an appropriate basis size for each problem. In order to free the users from this difficult task of selecting an appropriate basis size, we proposed an adaptive scheme (ν–TRLan) to dynamically determine the subspace dimension, which balances the expected computational cost and solution convergence rate, at every restart. An open source software package that implements both TRLan and ν–TRLan in C to solve Hermitian eigenvalue problems is available in the public domain.

Numerical results of synthetic problems have shown that ν–TRLan can not only automate the selection of the subspace dimension, but also improve the performance of TRLan that uses an optimal fixed basis size. To demonstrate the effectiveness of ν–TRLan in a real application, we applied it to the electronic structure calculations of quantum dots: namely, we replaced the state-of-the-art eigensolver in Parallel Energy Scan (PESCAN), which is based on the Preconditioned Conjugate Gradient (PCG) method, with ν–TRLan. We have demonstrated that when computing more than 30 eigenpairs, ν–TRLan performs significantly better than PCG.

Even though we have focused on TRLan in this paper, other subspace methods such as ARPACK [1, 8] and PRIMME [16, 17] also require a user to select an appropriate basis size for each problem. Our adaptive projection subspace dimension scheme can be applied to such subspace methods. In particular, the convergence analysis in Section 3.2 is directly applicable to ARPACK.

6 Acknowledgements

The authors gratefully acknowledge the helpful discussions and suggestions received from Drs. Osni Marques and Christof Vömel. This work was supported in part by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. 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.

References

[1] D. Calvetti, L. Reichel, and D. Sorensen, *An implicitly restarted Lanczos method for large symmetric eigenvalue problems.*, Electronic Transactions on Numerical Analysis, 2 (1994), pp. 1–21.

[2] A. Canning, L. W. Wang, A. Williamson, and A. Zunger, *Parallel empirical pseudopotential electronic structure calculations for million atom systems*, J. Comput. Phys., 160 (2000), pp. 29–41.

[3] M. Crouzeix, B. Philippe, and M. Sadkane, *The Davidson method*, SIAM J. Sci. Comput., 15 (1994), pp. 62–76.

[4] J. Cullum and R. A. Willoughby, *Lanczos algorithms for large symmetric eigenvalue Computations: Theory*, vol. 3 of Progress in Scientific Computing, Birkhauser, Boston, 1985.

[5] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[6] G. H. Golub and C. F. van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD 21211, 3rd ed., 1996.

[7] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Nat. Bur. Stand., 45 (1950), pp. 255–281.

[8] R. Lehoucq, D. Sorensen, and C. Yang, *ARPACK Users Guide: solution of large scale eigenvalue problems with implicitly restarted Arnoldi methods*, SIAM, Philadelphia, PA, 1998. ARPACK Software is available at `http://www.caam.rice.edu/software/ARPACK/`.

[9] J. Li and L.-W. Wang, *First principle study of core/shell structure quantum dots*, Applied Physics Letters, 84 (2004), pp. 2648–3650.

[10] P. B. Morgan, *On restarting the Arnoldi method for large nonsymmetric eigenvalule problems.*, Mathematics of Computation, 65 (1996), pp. 1213– 1230.

[11] B. N. Parlett, *The symmetric eigenvalue problem.*, Classics in Applied Mathematics. SIAM, Philadelphia, PA, 1998.

[12] M. C. Payne, M. P. Teter, D. C. Allan, T. A. Arias, and J. D. Joannopoulos, *Iterative minimization techniques for ab-initio total energy calculations: molecular dynamics and conjugate gradients*, Rev. Mod. Phys., 64 (1992), pp. 1045–1097.

[13] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, Manchester University Press, Manchester, UK, 1993.

[14] J. Schrier and L.-W. Wang, *A systematic first principles study of nanocrystal quantum-dot quantum wells*, Phys. Rev. B, 73 (2006).

[15] H. D. Simon, *The Lanczos algorithm with partial reorthogonalization*, Math. Comp., 42 (1984), pp. 115–142.

[16] A. Stathopoulos, *Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. Part I: Seeking one eigenvalue*, SIAM J. Sci. Comput., 29 (2007), pp. 481–514.

[17] A. Stathopoulos and J. R. McCombs, *PRIMME: Preconditioned iterative multimethod eigensolver*. `http://www.cs.wm.edu/~andreas/software/doc.pdf`, 2005.

[18] A. Stathopoulos, Y. Saad, and K. Wu, *Dynamic thick restarting of the Davidson and the implicitly restarted Arnoldi methods.*, SIAM J. Sci. Comput., 19 (1998), pp. 227–245.

[19] G. W. Stewart, *Adjusting the Rayleigh Quotient in semiorthogonal Lanczos methods*, SIAM J. Scient. Comput., 24 (2002), pp. 201–207.

[20] L. Wang and A. Zunger, *Solving Schrodinger's equation around a desired energy: Application to silicon quantum dots*, J. Chem. Phys., 100 (1994), pp. 2394–7.

[21] ——, *Pseudopotential theory of nanometer silicon quantum dots application to silicon quantum dots*, in Semiconductor Nanocluster, P. Kamat and D. Meisel, eds., 1996, pp. 161–207.

[22] K. Wu and H. Simon, *Thick-restart Lanczos method for large symmetric eigenvalue problems.*, SIAM J. Mat. Anal. Appl., 22 (2000), pp. 602–616.

[23] ——, *TRLan software package*. `http://crd.lbl.gov/~kewu/trlan.html`, 2000.

[24] ——, *TRLan user guide.*, Tech. Report LBNL-43178, Lawrence Berkeley National Laboratory, 2000.

[25] I. Yamazaki and K. Wu, *ν–TRLan software package project*. `https://codeforge.lbl.gov/projects/trlan/`.