1 Introduction

The field of quantum chemistry concerns itself with developing methods to solve the Schrodinger equation (SE) in order to extract material properties of interest, such as thermodynamic quantities and spectra. Two such methods have emerged as being truly ab initio in the sense that they do not rely on experimental data: the Full Configuration Interaction (FCI), and Coupled Cluster Theory. For the purposes of this discussion, we will primarily focus on CI, as it yields Hermitian matrices. The work herein focuses on the iterative diagonalization of Hermitian matrices using two common techniques in the field: the Lanczos and Davidson schemes.

However, it should be mentioned that this is only a simulation of working with these type of matrices as performing a CI-like calculation yields matrices that are orders of magnitude larger than those studied here. In fact, for a "small" molecule like benzene - having 6 carbon and 6 hydrogen atoms - it has been established that to perform a fully ab initio calculation requires on the order of 10^{35} Slater determinants; with matrices of similar size.(1) Corresponding matrices built from such a wavefunction have no hope of being stored in cache; many times they are stored on disk and read into memory block by block as needed. So at no point do we have such a large matrix in memory to work with; hence the need for clever diagonalization techniques if there is any hope of solving the Schrodinger equation for molecules larger than 3 or 4 atoms in an appreciable basis set. Coupled Cluster theory offers a high order polynomial-scaling route that rapidly converges to full CI results, yet still faces its own set of similar limitations.(2) Particularly in the context of this work, working on excited state spectra means dealing with non-Hermitian matrices which is outside the bound of the current discussion. It should also be noted that in the context of quantum chemistry, we are primarily only interested in a small subset of the corresponding eigenspectra for some large matrix A; typically the largest root in magnitude, or a few of the smallest roots in magnitude. Furthermore, regardless of which ab initio method is employed, a few definite things can be said about the resulting matrices that need diagonalization, namely that A is sparse and diagonally dominant.

2 Lanczos Algorithm

To begin, we will give a few ideas on the reasoning behind the Lanczos algorithm, and then describe why and how it works. Given some Hermitian matrix $A \in \mathbb{R}^{m\times m}$, where m is a large number $\geq 10^3$, we want to find a way to diagonalize said matrix that selectively targets a subset of the complete set of roots. The Lanczos algorithm can be stated as such: given A, an initial guess vector $v_1 \in \mathbb{R}^m$ and number of iterations n, find a tridiagonal $T \in \mathbb{R}^{n\times n}$ and unitary matrix $V \in \mathbb{R}^{m\times n}$ such that $T = V^*AV$. Note that because we are dealing with a Hermitian matrix A, putting the matrix T in upper-Hessenberg form is equivalent to a tridiagonal matrix. The hope then is to diagonalize the matrix T which is hopefully of much smaller dimension than A to approximate extremal eigenvalues of A. This gives us a rough outline on what we expect from the Lanczos algorithm, but it does give us a reason why this procedure works.

In this context, let's take the full example where the number of iterations is equivalent to the dimension of the matrix A. We expect in this case, that the Lanczos scheme should find a matrix T that, after diagonalization, are numerically equivalent to the roots of those arising from the full diagonalization of the starting matrix A. The proof of this is found in the following:

Proof: Suppose A is Hermitian, λ is an extremal eigenvalue of A we are searching for, a factorization $V^*AV = T$ has been found, and $Tx = \lambda x$. Given the corresponding eigenvector of A, $y = Vx$,

$$Ay = (VTV^*)y = (VTV^*)(Vx) = V(Tx) = \lambda(Vx) = \lambda y \tag{1}$$

So by diagonalizing T, we effectively find the eigenvalues of A. If I may, I will call this a similarity transformation since T and A do not necessarily share the same eigenvectors but do share the same eigenvalues.

So we have shown that by working with the matrix T of similar dimension as the starting matrix A, we can obtain the complete eigenspectra. But of course, this doesn't necessarily help us out, as we don't want to work with matrices of dimension A, and we don't necessarily need the complete eigenspectra - only a subset. This brings us to the point: the Lanczos procedure is an implementation of the Rayleigh-Ritz procedure on a sequence of Krylov subspaces. We define these terms formally in the following:

Rayleigh Quotient: In Quantum Chemistry, the idea of a Rayleigh quotient is frequently referred to as the Variational Principle. The ideas are essentially the same: determine a series of expectation values that are upper bounds to the "correct" roots of matrix A. In the current context, we use the Rayleigh quotient and $y \neq 0$ to generate v_k such that

$$r(y) = \frac{y^T A y}{y^T y} \rightarrow m_k = \min_{y \neq 0} \frac{x^T (V_k^T A V_k) x}{x^T x} = \min_{||x||_2 = 1} r(V_k x) \geq \lambda_n(A) \tag{2}$$

where m_k is an increasingly better approximation to λ upon successive iteration.

Krylov Subspace: The Krylov matrix $K^m(v)$ are given to be $K^m(v) \equiv (v, Av, \cdots, A^{m-1}v)$, where the Krylov subspace is essentally the span of such a matrix. The Lanczos basis is then defined as the columns $K^m(v)$ after modification by the Gram-Schmidt procedure; hence the columns are orthonormal.

Using the notes from the course as a reference, and a little algebra on the equation $AV = VT$, we are finally prepared to give some pseudocode for what a Lanczos iteration actually looks like. Notice that the nth iteration can be written as $Av_n = \sum_{j=1}^{n+1} t_{jn} v_n$. If we expand this out, we can determine what the column vector for the $n+1$ iteration looks like: $v_{n+1} = (Av_n - \sum_{j=1}^{n} v_j t_{jn}) \frac{1}{t_{n+1,n}}$, where in general $t_{jn} = v_j^* (Av_n)$ due to the orthogonality of the columns of V. In effect, this procedure generates an othonormal basis in a Krylov subspace. The original paper, starting on page 266, offers a very nice explanation actually.(3) To be clear equating columns k of $AV = VT$ and using a little algebra , we see that for iteration $k = 1, \cdots, n-1$

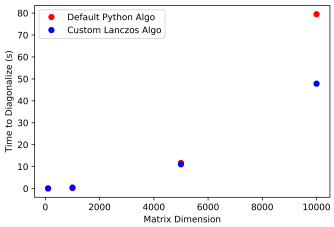

Figure 1: Timings for full Diagonalization

$$Av_k = \beta_{k-1} v_{k-1} + \alpha_k v_k + \beta_k v_{k+1} \tag{3}$$

where we can naturally solve for v_{k+1} and use this as our guess vector in the following iteration. Projecting this on to v_k and using the fact that the columns of V are orthonormal shows that $\alpha_k = v_k^T A v_k$, $\beta_k = v_{k+1}^T A v_k$ for scalars α, β.

Implementation: If choose a random starting $r_0 = v_0$, we can define a vector $r_k = (A - \alpha_k I) v_k - \beta_{k-1} v_{k-1}$ where $v_{k+1} = \beta_k^{-1} r_k$ and $\beta_k = ||r_k||_2$. We iterate this up to n times, or until $r_k = 0$. This implies that our signal for convergence is when $AV_k - V_k T_k = r_k e_k^T$ or $\beta_k = 0$. In the limit $k \rightarrow \infty$, $\alpha_k = v_k^T A^k v_k \approx \lambda$.

The way I have chosen to write the algorithm, I construct T and V at the end of each iteration, such that

$$V = \begin{pmatrix} v_1 & v_2 & \cdots & v_m \end{pmatrix} \tag{4}$$

$$T = \begin{pmatrix} \alpha_1 & \beta_1 & 0 & \cdots & 0 \\ \beta_2 & \alpha_2 & \beta_3 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ & & & & \end{pmatrix} \tag{5}$$

at which point I perform a diagonalization on T, sort the resulting eigenvalues, and determine if the lowest root of that iteration is different up to some tolerance of the prior iterations lowest root.

Now that we have introduced the foundation for why and how this algorithm works, I will discuss the implementation. As spacing is limited in this report to see the actual algorithm, please see the jupyter-notebook implementation or powerpoint snapshot for the actual program. Note that Figures 1-3 intend to study the algorithm without any stopping criteria other than some maximum iteration, which is general was chosen to be ≈ 50 iterations.

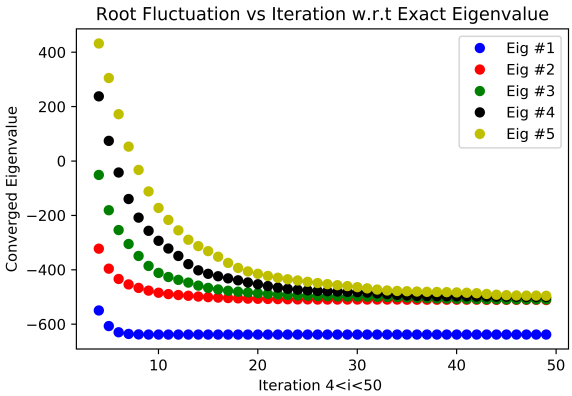

Figure 2: Convergence of 5 lowest eigenvalues

Figure 1 details the algorithm's time performance of full diagonalization of matrices with increasing dimension as compared to default Numpy routines. As can be seen, an iterative technique based on Lanczos algorithm performs rather well up until the dimension reaches about 10,000, in which case performance drops rapidly. This was done out of curiosity, as we have already established we are not interested in using this method to obtain the full eigenspectra; only a subset. Figure 2 shows the convergence of the 5 lowest roots with respect to iteration. As can be seen, Lanczos convergences rapidly to the lowest root, but takes additional cycles to converge sequentially larger roots. I believe this to be an artifact of our Variational Principle. Figure 3 simply shows that Lanczos converges within 10 iterations to the lowest root; this is relatively quick in terms of CPU cycles.

3 Block Davidson Algorithm

In my limited experience, it would seem that the Lanczos scheme has been eclipsed by block Davidson in much of quantum chemistry. The exceptions are typically fringe cases (an area I am currently working in) which won't be discussed. Furthermore, much of the foundation behind Davidson's method has already been laid in the section detailing Lanczos' algorithm. In fact, in some sense Davidson's method can be thought of as a preconditioned version of the Lanczos method that uses the Rayleigh-Ritz procedure but the use of the Krylov subspace is somewhat disputed in the literature.(4) From my knowledge, it seems like quantum chemistry uses a Davidson algorithm that employs the Krylov subspace.

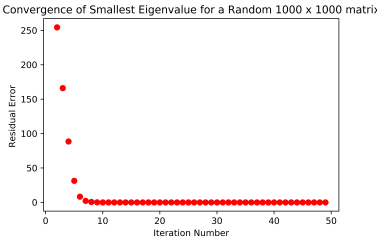

Figure 3: Convergence of Lanczos' Algorithm for lowest root

As these ideas have already been detailed, I will simply introduce the primary differences (of which there are 3) between the Block Davidson and Lanczos algorithms. The first major difference is that this algorithm allows for user specified number of initial guess vectors. The second major difference is that a QR factorization is needed at the beginning of each iteration to orthonor-

malize the columns of V, as opposed to the Gram-Schmidt like orthogonalization that Lanczos employs. To reiterate, the third major difference is that the Davidson algorithm employs a shift to subsequent guess vectors.

Figure 4 defines the form of the starting Hermitian matrix A, for both Lanczos and Davidson. We see that we can "tune" the degree to which this matrix is dense; if this factor is 1, this implies that the matrix is dense. A factor less than 1 implies a degree of sparsity exists in the matrix. Much of this algorithm's use in the field of quantum chemistry relies on the starting matrix being sparse and diagonally dominant.

```
sparsity = 0.0001
Amat = np.zeros((dim,dim))
for i in range(0,dim):
    Amat[i,i] = i + 1
Amat = Amat + sparsity*np.random.randint(20,size=(dim,dim))
Amat = (Amat.T + Amat)/2
```

Figure 4: Example of Matrix Sparsity

Figure 5 depicts the Davidson algorithm's convergence according to the number of initial guess vectors. As can be seen, the larger the number of initial guess vectors, the faster the convergence of the lowest root. In practice, we typically ask for twice as many roots as we actually need. Obviously there is a speed boost from this. However, I have noticed that if I ask for the exact number of roots I need, sometimes the largest root may not be in the correct sequential order as compared to a situation where I asked for double the number of roots. I have not been able to recreate this in my code yet, but I know this problem exists in quantum chemistry and represents a significant hurdle as out of order roots impact accuracy of the prediction.

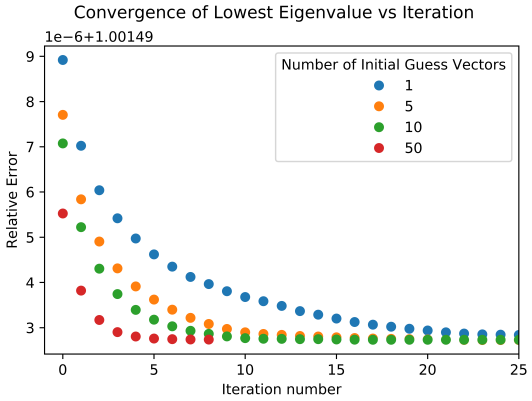

Figure 5: Convergence of Smallest eigenvalue w.r.t. number of initial guess vectors

Figure 6 illustrates the efficiency of the Davidson scheme, assuming there is a high degree of sparsity in the starting matrix; otherwise, Davidson has to perform more iterations to converge the lowest root. Sparsity factors between 0.1 and 0.01 exhibit tremendous change in terms of error with respect to iteration number. Sparsity factors lower than this converge so fast, the plot shown here has trouble resolving the dots as they all lie on top of one another. It should be noted this figure does not have any stopping criteria associated with it, and ran for roughly 150 iterations.

4 Comparison of Davidson, Lanczos, and Default Numpy Diagonalization

As a final test of my implementations for the Lanczos and block Davidson algorithms, I decided to compare against default Numpy diagonalization routines for accuracy and time. Figures 7 and 8 depict this. As can be seen, Davidson's scheme using 50 initial guess vectors finds the lowest eigenvalue the fastest, although Lanczos is also rather fast. In fact, both of these algorithm's find the lowest root of a 1000 by 1000 dimension matrix A faster than default Numpy can find all the roots. However, when comparing the accuracy of the 5 smallest eigenvalues, it was found that Lanczos immediately incurs a 0.02 error for the lowest root. This error gets exasperated for larger roots as well. On the other hand, Davidson's method remains relatively accurate when comparing all 5 roots. The stopping criteria was set such that a current iteration's lowest eigenvalue is below 0.01 larger than the previous iteration's lowest eigenvalue for both methods. This is an admittedly pretty relaxed criteria, and could be tightened up quite a bit to improve numerical results. However, as it stands, I have shown that the currently

implemented versions of the Lanczos and Davidson algorithms are valuable as proofs of concept since I do not necessarily have to rely on built in python routines to fully diagonalize a matrix if I only want a subset of the eigenspectra and the matrix is both sparse and Hermitian.

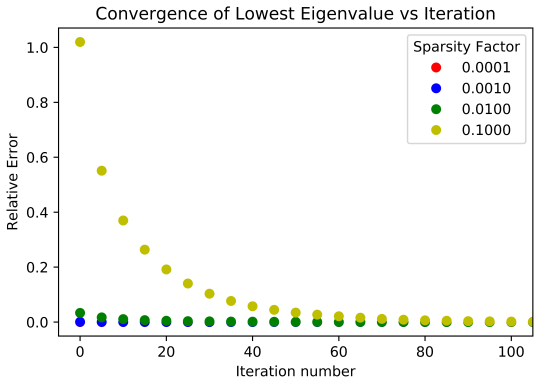

Figure 6: Convergence of Smallest eigenvalue w.r.t the sparsity of initial matrix

```
                ** Time Comparison (seconds) **
Davidson (1 initial guess vector):       2.4154858589172363
Davidson (5 initial guess vector):       2.039685010910034
Davidson (10 initial guess vector):      1.185352087020874
Davidson (50 initial guess vector):      0.31585216522216797
Lanczos:                                 0.5464720726013184
Default Numpy:                           0.7308790683746338
```

Figure 7: Time Comparison

5 Conclusion

```
                                ** Comparison of first 5 eigenvalues **
Default Numpy:                          [1.07046677 2.1063181  3.00166359 4.16139663 5.13262811]
Davidson (50 initial guess vectors):    [1.07862717 2.11842194 3.01705597 4.17411521 5.14386712]
Lanczos:                                [ 1.05260919  3.04001205  5.13205332  8.86525584 11.30008985]
```

Figure 8: Accuracy Comparison

Although much work has been done, I feel like this has only been a brief introduction to this field. More time will be focused on non-Hermitian analogs of the block Davidson scheme, which is pertinent to my work. At some point I will also tackle the original 86 paper detailing Davidson's algorithm, but in the time allotted I honestly could not understand what was being said. Also note I have only attached "cleaned" version of the programs to perform Lanczos and Davidson; the dirty version I have performs all the required plotting. I can supply it upon demand.

References

[1] Eriksen JJ, Anderson TA, Deustua JE, Ghanem K, Hait D, Hoffmann MR, et al. The ground state electronic energy of benzene. The Journal of Physical Chemistry Letters. 2020;11(20):8922–8929.

[2] Bartlett RJ, Musiał M. Coupled-cluster theory in quantum chemistry. Reviews of Modern Physics. 2007;79(1):291.

[3] Lanczos C. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. United States Governm. Press Office Los Angeles, CA; 1950.

[4] Jenkins HK. Eigenvalue problems from electronic structure theory. 1998;.