Customized Sparse Eigenvalue Solutions in Lighthouse

Ramya Nair
University of Colorado Boulder, ramyakk17@gmail.com

Follow this and additional works at: https://scholar.colorado.edu/csci_gradetds

Part of the Numerical Analysis and Scientific Computing Commons

Recommended Citation
Nair, Ramya, "Customized Sparse Eigenvalue Solutions in Lighthouse" (2014). Computer Science Graduate Theses & Dissertations. 89.
https://scholar.colorado.edu/csci_gradetds/89

This Thesis is brought to you for free and open access by Computer Science at CU Scholar. It has been accepted for inclusion in Computer Science Graduate Theses & Dissertations by an authorized administrator of CU Scholar. For more information, please contact cuscholaradmin@colorado.edu.
Customized Sparse Eigenvalue Solutions in Lighthouse

by

Ramya Nair

B.Tech., Kerala University, India, 2005

A thesis submitted to the
Faculty of the Graduate School of the
University of Colorado in partial fulfillment
of the requirements for the degree of
Master of Science
Department of Computer Science
2014
This thesis entitled:
Customized Sparse Eigenvalue Solutions in Lighthouse
written by Ramya Nair
has been approved for the Department of Computer Science

__________________________
Elizabeth Jessup

__________________________
Boyana Norris

__________________________
Xiao-Chuan Cai

Date ________________

The final copy of this thesis has been examined by the signatories, and we find that both the
content and the form meet acceptable presentation standards of scholarly work in the above
mentioned discipline.
We come across sparse eigenvalue problems in many areas of scientific computing. A variety of high-performance numerical software packages with many different eigensolvers are available to solve such problems. Two main challenges arise while solving these problems: 1) finding the eigensolver routine that can correctly solve the problem and 2) implementing the desired solution accurately and efficiently using the appropriate software package. Our work overcomes these two challenges by intelligently identifying the sparse eigensolvers that are likely to perform the best for the given user-specified input characteristics and auto generating customized code template that uses the identified best solver. Lighthouse Taxonomy is a novel interface and search platform for users seeking software solutions to linear algebra problems. For this thesis, we integrated SLEPc into Lighthouse to support sparse eigenvalue solutions.
Acknowledgements

I would like to express my deepest gratitude to my advisor, Professor Elizabeth R Jessup, for giving me the opportunity to work on this research; it has been a life changing experience. Guidance and continuous support provided by her during the course of this research will stay with me for life.

My sincere thanks to Boyana Norris for helping me with out-of-the-box ideas and methods to try for this research. I got to learn a lot from these experiences.

My heartfelt thanks to Sa-Lin Cheng Bernstein, who provided me with continuous encouragement and valuable inputs throughout my thesis especially with the UI design.

I thank Professor Xiao-Chuan Cai, for piquing my interest in eigenvalue solutions by giving me opportunity to work with his research team for a visualization project in fall 2012.

I also thank my fellow research associates, Javed Hussain and Luke Groeninger. Working with them has been a pleasure and an unforgettable learning experience.

Last but not the least, I thank my husband, my parents and my sister without their support none of this was possible.

This work utilized the Janus supercomputer, which is supported by the National Science Foundation (award number CNS-0821794) and the University of Colorado Boulder. The Janus supercomputer is a joint effort of the University of Colorado Boulder, the University of Colorado Denver and the National Center for Atmospheric Research.
Contents

Chapter

1 Introduction ................................................................. 1
   1.1 Problem Statement ................................................... 1
   1.2 Current Approach to the Solution ................................. 3
   1.3 Proposed Approach .................................................. 4

2 Background ................................................................. 5
   2.1 Lighthouse Taxonomy ................................................. 5
      2.1.1 User-Interface .................................................. 6
      2.1.2 Back-end implementation ..................................... 10
      2.1.3 Supporting sparse eigensolvers ............................... 11
   2.2 PETSc ................................................................. 11
   2.3 SLEPc ................................................................. 12
      2.3.1 Why SLEPc ...................................................... 12
      2.3.2 SLEPc components ............................................. 14
      2.3.3 SLEPc eigensolver routines .................................. 15
   2.4 Machine Learning .................................................... 17
      2.4.1 Introduction ..................................................... 17
      2.4.2 Decision tree ................................................... 18
      2.4.3 Support vector machine ...................................... 20
      2.4.4 Machine learning applications ............................... 22
3  Research Approach  
   3.1  Finding the right questions ................................................. 23  
   3.2  Finding the right SLEPc solver ........................................... 24  
   3.3  Integration with Lighthouse ................................................ 24  

4  SLEPc Eigensolver Integration  
   4.1  The Feature Set .................................................................... 25  
   4.2  Experimental setup ............................................................. 27  
      4.2.1  Test bed ................................................................. 28  
      4.2.2  Analysis of SLEPc implementation ................................. 28  
      4.2.3  SLEPc code for eigensolver ......................................... 30  
      4.2.4  Script file generation for experiments ............................ 31  
   4.3  Running the experiments ..................................................... 32  
      4.3.1  Data .................................................................. 32  
      4.3.2  Experiments ............................................................ 32  
   4.4  Training and Prediction ....................................................... 32  
      4.4.1  Finding optimum solver (Training data setup) .................. 35  
      4.4.2  Intelligent solver selection (Prediction) ........................... 37  
   4.5  Result validation ............................................................... 41  
      4.5.1  Decision tree classification in MATLAB .......................... 42  
      4.5.2  Decision tree classification in Orange ............................. 43  
      4.5.3  Support vector machine in Orange ..................................... 44  
   4.6  Result Analysis .................................................................. 45  
      4.6.1  Eigensolver vs matrix properties ..................................... 45  
      4.6.2  Spectrum vs Problem type .......................................... 46  
      4.6.3  Numeric property plots ............................................... 47  
   4.7  Integration with Lighthouse .................................................. 48
### Tables

#### Table

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Sample input structure to Decision tree algorithm</td>
<td>18</td>
</tr>
<tr>
<td>4.1</td>
<td>Janus supercomputer details</td>
<td>28</td>
</tr>
<tr>
<td>4.2</td>
<td>Sparse matrix files used in experiments</td>
<td>33</td>
</tr>
<tr>
<td>4.3</td>
<td>Input parameters tested</td>
<td>34</td>
</tr>
<tr>
<td>4.4</td>
<td>Sample test case from sparse matrix file shipsec1.mtx</td>
<td>35</td>
</tr>
<tr>
<td>4.5</td>
<td>Sample results from sparse matrix file shipsec1.mtx for input case in table 4.4</td>
<td>35</td>
</tr>
<tr>
<td>4.6</td>
<td>Sample test case with rank from sparse matrix file shipsec1.mtx</td>
<td>38</td>
</tr>
<tr>
<td>4.7</td>
<td>Decision tree cross validation results from MATLAB</td>
<td>42</td>
</tr>
<tr>
<td>4.8</td>
<td>Decision tree cross validation results from Orange</td>
<td>43</td>
</tr>
<tr>
<td>4.9</td>
<td>SVM cross validation results from Orange</td>
<td>44</td>
</tr>
</tbody>
</table>
Figures

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Lighthouse basic structure of the user-interface</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>Lighthouse guided search for LAPACK linear solver routines.</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Lighthouse advanced search for LAPACK linear solver routines.</td>
<td>9</td>
</tr>
<tr>
<td>2.4</td>
<td>Lighthouse keyword search for LAPACK linear solver routines.</td>
<td>10</td>
</tr>
<tr>
<td>2.5</td>
<td>Organization of PETSc libraries</td>
<td>11</td>
</tr>
<tr>
<td>2.6</td>
<td>Softwares providing eigenvalue solutions</td>
<td>13</td>
</tr>
<tr>
<td>2.7</td>
<td>Components of SLEPc and PETSc</td>
<td>14</td>
</tr>
<tr>
<td>2.8</td>
<td>Sample decision tree predicting a computer purchase</td>
<td>18</td>
</tr>
<tr>
<td>2.9</td>
<td>Decision tree partitioning mechanism</td>
<td>20</td>
</tr>
<tr>
<td>2.10</td>
<td>SVM maximum margin hyperplane</td>
<td>21</td>
</tr>
<tr>
<td>4.1</td>
<td>Part of the generated classification tree in MATLAB</td>
<td>39</td>
</tr>
<tr>
<td>4.2</td>
<td>Part of the generated classification tree in ORANGE</td>
<td>40</td>
</tr>
<tr>
<td>4.3</td>
<td>Cross-validation technique</td>
<td>42</td>
</tr>
<tr>
<td>4.4</td>
<td>Analyzing eigensolver vs portion of spectrum</td>
<td>45</td>
</tr>
<tr>
<td>4.5</td>
<td>Analyzing eigensolver vs binary</td>
<td>46</td>
</tr>
<tr>
<td>4.6</td>
<td>Analyzing portion of spectrum vs problem type</td>
<td>47</td>
</tr>
<tr>
<td>4.7</td>
<td>Analyzing size vs tolerance</td>
<td>48</td>
</tr>
<tr>
<td>4.8</td>
<td>Analyzing number of eigenvalues vs tolerance</td>
<td>49</td>
</tr>
</tbody>
</table>
4.9 SLEPc guided search UI (under development)
Sparse eigenvalue problems are increasingly being seen with the emergence of big data and high performance computing. We see such problems appearing in scientific fields ranging from subatomic particle theories in quantum physics [25] to structural engineering [10]. Its widening usage in the increasingly connected internet era is well known in, for example, search engines [11] and social networks[34].

Extensive research in numerical linear algebra has given us different kinds of solutions to eigenvalue problems and most of these solutions optimize depending on the matrix properties or the results sought. Hence, when practitioners in these various fields encounter eigenvalue problems, they seek the best solution with respect to a specific set of input conditions. As the input changes, a different ‘best’ solution is sought.

To cater to the needs of the increasing demand of solutions to such varied eigenvalue problems, a number of free and licensed software packages are available. These packages cover many different kinds of eigenvalue solutions and somewhere in that pile of solutions, lies one solver that most accurately and efficiently solves the particular problem that the user seeks.

1.1 Problem Statement

Solving a sparse eigenvalue problem accurately and efficiently takes significant effort and may require knowledge in applied mathematics, computational methods, and the problem domain. First of all, knowledge of the various methods that solve eigenvalue problems is important. Some
methods or algorithm may converge much faster than others for the particular problem at hand. One needs to have thorough knowledge of numerical linear algebra and extensive experience solving eigenvalue problems to make a good judgement. Inappropriate solver selection and configuration may lead to unwanted results such as a high residual or even no convergence at all.

The writing of accurate and efficient implementations of algorithms is beyond the capabilities of most practitioners and programmers. A numerical linear algebra expert who decides on a particular solver may not have enough programming expertise to develop an efficient implementation, especially if a parallel solution is vital. Similarly, even an experienced programmer may lack the applied mathematics background required to implement the sparse eigenvalue solution. Hence, it is common for the non-expert to turn instead to software packages. The selection of an appropriate software package depends on the resources at one’s disposal ranging from hardware criteria, such as the number of processors available, to personal convenience such as the preferred software language.

When the decisions about the appropriate solver and software package are made, the potentially enormous task of correct and efficient implementation using the selected software packages still remains. A good result requires not only general programming expertise but also a sound understanding of the selected software package framework and considerable skill in producing efficient parallel implementations.

Hence, one requires proficiency in multiple fields to obtain a satisfactory solution. The challenge, therefore, is selecting the best solution from those available and implementing and optimizing the code. As a result, obtaining a desirable solution can be a very taxing procedure for a person not experienced with linear algebra, programming, parallel implementation and the different software packages available for sparse eigenvalue problems. Thus, it is a common scenario that deciding on and implementing a good solution for the sparse eigenvalue problem may take a considerable amount of time and effort.
1.2 Current Approach to the Solution

As explained in the previous section, the first step is to find the algorithm or method that can solve the sparse eigenvalue problem encountered. For a person with expertise in eigenvalue problems, the answer to this question may be at his/her fingertips. For others, finding an acceptable method may require some research. Going through web search engines or Wiki articles may be the first step for a novice to get some basic idea on the topic. However, search engines are of limited utility because they do not provide well-organized results for linear algebra solutions, and a user may have to go through a number of links before coming across a useful one. In addition, it may be necessary for the user to consult a number of books or papers to understand the implementation and properties of various eigensolver algorithms so as to match the eigenproblem to the best eigensolver. After thorough analysis of these methods, the user may be able to select an algorithm that works for the particular problem at hand.

For the next task of finding a fitting software package, the user needs to sort through the many different packages available and learn enough about each of them to be able to make a good choice. For full understanding, the user may need to go through the manual pages of the selected package thoroughly. To be able to appropriately use the features of the eigenvalue routine selected in the previous step, the user has to spend considerable time to understand the package framework.

The next task is the unavoidable one of implementing the selected solver using the chosen software package. The numerical linear algebra expert who decides on the eigenvalue solver may not be the most proficient in programming hence the task of coding may be delegated to a programmer, who on the other hand may be inexperienced in numerical computing. Either way, it requires some learning curve on both ends. Anyone who decides to program will now have to go through the entire software package manual before writing code.

Even after rigorous comparative analysis of the methods and careful selection of software packages, the code generated may or may not work for the particular problem at hand depending on various factors. Even when using high-quality software, an unsatisfactory solution may still
result. For example, the number of converged eigenvalues may be different from expected or the residual may be greater than expected. Even if no mistakes are made in any of the steps explained above, the chosen solver may fail to produce the desired solution for the given problem, and it may be necessary to repeat some or all of the development steps.

1.3 Proposed Approach

This thesis proposes to solve the problems stated above by providing an application that provides a convenient interface such that the user can make selections to specify information about the sparse eigenvalue problem. The application then finds the most favorable solvers, generates code for the specific problem using a known software package and provides it to the user. The user can then download and plug it into desired application for use.

To provide the described functionality to the users, we first select an appropriate software package which solves sparse eigenvalue problems. We then experiment with different matrices and input parameters and note some of the characteristics of the result for each of the applicable solver. Next, we algorithmically determine the solvers that achieves the best results. To suggest an appropriate solver for any user specified problem, we make use of machine learning techniques so as to create a model using the results obtained from these experiments. This model can intelligently determine the solver that will achieve the best results for any sparse eigenvalue problems. The generated model is then integrated into Lighthouse taxonomy, a web interface that helps users to find solutions to linear algebra problems. The Lighthouse taxonomy is explained in chapter 2.

Integrating sparse eigenvalue solutions into Lighthouse taxonomy will be useful to those who seek eigenvalue solutions in different ways. It may help the linear algebra experts in programming efficient code, the programmers in intelligently finding the best eigenvalue solvers and the scientific domain experts in the entire solution process itself. Such an application will not only save a lot of time in finding eigenvalues but will also make complicated solutions much more accessible to a larger user base. Users can, hence, spend time on the actual scientific computation problem rather than worrying about finding and implementing an optimum solver.
Chapter 2

Background

This chapter details on some of the topics that can help understand the different steps involved in this research. In the first section, we describe the web interface Lighthouse taxonomy that helps users to find solutions to linear algebra problems. We detail on current supported functionalities, the user interface and basic back-end implementation. We also brief on how the support for sparse eigenvalue problems will be provided through Lighthouse.

We use the software package SLEPc [24], which contains a variety of iterative sparse eigenvalue solvers, to add to the taxonomy in the Lighthouse application. SLEPc is built on top of PETSc [6], which provides a set of tools and base data structures for advanced numerical problems. The next two sections explains the two software packages and its usage in this research.

Finally, we elaborate on some of the machine learning concepts which acts as the back bone of this research. We describe some powerful classification techniques that helped us in creating a model that can intelligently determine the best eigensolver for different user scenarios.

2.1 Lighthouse Taxonomy

Lighthouse is an open source web application that provides users with the best solutions for their linear algebra computations [26]. It acts as an interface for users to perform customized searches for a variety of numerical problems. The taxonomy is an effective classification of existing linear algebra software presently including LAPACK [5] and PETSc [6]. Different search options allows the user to make informed selection from the filtered set of the routines that solves the user
Lighthouse can be used to create code templates in FORTRAN 90 and C containing working programs that declare arguments and call selected routines. Users can download and modify the templates to meet their particular project needs. Furthermore, Lighthouse provides the ability to automatically generate and tune high-performance implementations of custom linear algebra computations by interfacing with the Build to Order (BTO) [30] compiler. BTO produces highly tuned C implementations based on high-level MATLAB-like input specification of the computation.

LAPACK software package is integrated with Lighthouse to add support for solutions to dense linear equations and dense eigenvalue problems. The large number of LAPACK routines can be easily searched through Lighthouse interface to find the particular routine that most fits the user specified problem.

Apart from providing the building blocks for numerical problems (refer to section 2.2), the PETSc software package also consists of solutions to sparse linear equations. Adding PETSc to the Lighthouse taxonomy lets the user search for solution to sparse linear equations and generate the respective code template.

2.1.1 User-Interface

This section shows and describes the basic structure of the Lighthouse user interface. The Lighthouse UI can be divided into three parts, the search, the result and the work area. The figure 2.1 distinguishes these areas.

The search area provides different ways for the user to perform customized searches. Lighthouse provides three kinds of search options: guided search, advanced search and keyword search. The result area presents to the user a filtered set of routines that fit the current search selections. The work area is used as the final step where user can drag the routines of interest from the filtered set in result section into the work area. User can then generate and download the code templates for the selected routine. These steps are shown in figure 2.1.
Figure 2.1: Lighthouse basic structure of the user-interface

- **Lighthouse Search**: Provides customized search options
- **Lighthouse Results**: Filtered set of routines for users search selection
- **Lighthouse Work Area**: Customized code generation for selected routines
In the following subsections we detail the three search approaches in Lighthouse.

2.1.1.1 Guided search

The guided search is designed to lead users through increasingly refined routine searches until the desired result is attained. Fig. 2.2 illustrates the Lighthouse guided search for LAPACK linear solver routines. Note that in the guided search section, user is prompted with a series of simple to understand questions about their problem domain. On answering these questions, appropriate set of solver routines are selected from the database and displayed in the results section of the interface. The user can now drag and drop the routine(s) into the work area and generate code in the desired language. The code is displayed on the right side of the work area and can be downloaded for direct use.

![Figure 2.2: Lighthouse guided search for LAPACK linear solver routines.](image)
2.1.1.2 Advanced search

The advanced search is recommended for users who are familiar with the library. Figure 2.3 shows a sample advanced search selection. For this search option, user must be aware of some of the basic terminologies with respect to the library and the linear algebra problem. For example, as shown in the figure 2.3, user must know the meaning of computational routines (vs driver routines) in LAPACK. The result section for advanced search selection displays the routines in a tabular format, specifying the property of each routine. The work area has similar functionality as explained in guided search section.

![Lighthouse advanced search for LAPACK linear solver routines.](image)

2.1.1.3 Keyword search

All users can benefit from the keyword search, which allows for routine search through an input keyword or phrase. Figure 2.4 shows a sample keyword search for the keywords ‘solving a
system of linear equations hermitian’. This search resulted in six LAPACK routines. The result and the work area behaves similar to that detailed in guided search (refer to section 2.1.1.1)

![Figure 2.4: Lighthouse keyword search for LAPACK linear solver routines.](image-url)

2.1.2 Back-end implementation

The Lighthouse web interface is built using the Django [7] which is an open source web application framework. Django is a python framework which can be used to develop well designed web interfaces rapidly. Its elegant design supports a variety a backend databases. For Lighthouse Taxonomy we use MySQL [32] database. All the routines in the Lighthouse supported software packages are loaded on to a well organized table structure for easy access. As the user makes selections during their search, an appropriate query to this database retrieves the respective solvers. For guided search, the retrieved solver routines are refined with every selection user makes.

Adding another software package is just a matter of adding respective table structure, loading data corresponding to the added library and setting up the views for the supported search options.
2.1.3 Supporting sparse eigensolvers

In this research we work on supporting sparse eigensolvers by adding the routines available in SLEPc software package into Lighthouse. We use machine learning techniques explained in section 2.4 to obtain the appropriate solvers for different input scenarios. Next, we collate these results and load it into the data tables created in MySQL. We then implement respective views to support various search options. These steps are explained in detail in the section 4.7.

2.2 PETSc

To understand the SLEPc package for sparse eigenvalue solvers a brief knowledge of PETSc package is necessary. Hence, this section gives an overview of PETSc before elaborating on the functionalities provided by SLEPc in the next section. PETSc stands for Portable, Extensible Toolkit for Scientific Computation and provides a set of building blocks which can be used to develop applications in order to solve numerical problems using parallel computing. It provides a powerful set of tools and basic data structures to ease the implementation of advanced numerical solutions.

![Figure 2.5: Organization of PETSc libraries](image-url)
Figure 2.5 [9] shows the basic organization of the libraries in PETSc [9]. Note that PETSc provides support ranging from simple data structures such as vectors and matrices to complex functionalities such as non linear solvers, pre-conditioners and time steppers. It uses the MPI standard to support parallel implementation of large scale applications.

2.3 SLEPc

As mentioned in the previous sections, to support sparse eigensolvers we add the solvers available in the SLEPc package into Lighthouse Taxonomy. This section elaborates the reason for this choice and details on the main features of the eigensolvers available in SLEPc.

2.3.1 Why SLEPc

There are a number of freely available software tools to solve large eigenvalue problems. We would like to select the one from which we can leverage a variety of eigenvalue solutions. Please find in figure 2.6 [22] list of all such freely available software tools.

Note from figure 2.6 [22] that SLEPc [12] supports a higher number of sparse eigenvalue solvers as compared to other solvers. Apart from the ones mentioned in the table, SLEPc also provides some additional solvers including Generalized Davidson, Jacobi Davidson and Rayleigh quotient conjugate gradient. The eigenvalues provided by SLEPc supports real as well as complex problems. Most of the SLEPc solvers work for Hermitian as well as non-Hermitian matrices. SLEPc also supports generalized eigenvalue solutions.

Hence, using SLEPc to support solutions to sparse solvers in Lighthouse is beneficial because the varied features provided by SLEPc covers a much broader range of solutions than other software packages.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Version</th>
<th>Date</th>
<th>Language</th>
<th>Par</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANASAZI</td>
<td>Block Krylov-Schur, block Davidson, LOBPCG</td>
<td>9.0</td>
<td>2008</td>
<td>C++</td>
<td>M</td>
</tr>
<tr>
<td>BLKLAN</td>
<td>Block Lanczos [P]</td>
<td>-</td>
<td>2003</td>
<td>C/Matlab</td>
<td>-</td>
</tr>
<tr>
<td>BLOPEX</td>
<td>LOBPCG</td>
<td>-</td>
<td>2004</td>
<td>C/Matlab</td>
<td>M</td>
</tr>
<tr>
<td>BLZPACK</td>
<td>Block Lanczos [P+S]</td>
<td>04/00</td>
<td>2000</td>
<td>F77</td>
<td>M</td>
</tr>
<tr>
<td>EIGIFP</td>
<td>Inverse-free Krylov subspace method</td>
<td>2.1.1</td>
<td>2004</td>
<td>Matlab</td>
<td>-</td>
</tr>
<tr>
<td>IETL</td>
<td>Power, RQI, Lanczos [N]</td>
<td>2.2</td>
<td>2006</td>
<td>C++</td>
<td>-</td>
</tr>
<tr>
<td>INSYLAN</td>
<td>Indefinite Symmetric Lanczos</td>
<td>1.0</td>
<td>2000</td>
<td>Matlab</td>
<td>-</td>
</tr>
<tr>
<td>IRBLEIGS</td>
<td>Block Lanczos (implicit restart)</td>
<td>1.0</td>
<td>2002</td>
<td>Matlab</td>
<td>-</td>
</tr>
<tr>
<td>JADAMILU</td>
<td>Jacobi-Davidson (symmetric)</td>
<td>2.0</td>
<td>2009</td>
<td>F77</td>
<td>-</td>
</tr>
<tr>
<td>JDCG</td>
<td>Jacobi-Davidson (symmetric)</td>
<td>2.0</td>
<td>2009</td>
<td>F77</td>
<td>-</td>
</tr>
<tr>
<td>MPB</td>
<td>Conjugate Gradient, Davidson</td>
<td>1.4.2</td>
<td>2003</td>
<td>C</td>
<td>M</td>
</tr>
<tr>
<td>PDACG</td>
<td>Deflation-accelerated Conjugate Gradient</td>
<td>-</td>
<td>2000</td>
<td>F77</td>
<td>M</td>
</tr>
<tr>
<td>PRIMME</td>
<td>Block Davidson, JDQMR, JDQR, LOBPCG</td>
<td>1.1</td>
<td>2006</td>
<td>C/F77</td>
<td>M</td>
</tr>
<tr>
<td>PROPACK</td>
<td>SVD via Lanczos [P]</td>
<td>2.1/1.1</td>
<td>2005</td>
<td>F77/Matlab</td>
<td>O</td>
</tr>
<tr>
<td>PySPARSE</td>
<td>Jacobi-Davidson (symmetric)</td>
<td>1.0.1</td>
<td>2007</td>
<td>Python</td>
<td>-</td>
</tr>
<tr>
<td>SLEPc</td>
<td>Krylov-Schur, Arnoldi, Lanczos, RQI, Subspace</td>
<td>3.0.0</td>
<td>2009</td>
<td>C/F77</td>
<td>M</td>
</tr>
<tr>
<td>SPAM</td>
<td>Davidson with SPAM</td>
<td>-</td>
<td>2001</td>
<td>F90</td>
<td>-</td>
</tr>
<tr>
<td>TRLAN</td>
<td>Lanczos (dynamic thick-restart)</td>
<td>-</td>
<td>2006</td>
<td>F90</td>
<td>M</td>
</tr>
<tr>
<td>ARNCHEB</td>
<td>Arnoldi-Chebyshev</td>
<td>-</td>
<td>1997</td>
<td>F77</td>
<td>-</td>
</tr>
<tr>
<td>ARPACK</td>
<td>Arnoldi/Lanczos (implicit restart)</td>
<td>2.1</td>
<td>1995</td>
<td>F77</td>
<td>M</td>
</tr>
<tr>
<td>ARPACK++</td>
<td>Arnoldi/Lanczos (implicit restart)</td>
<td>1.1</td>
<td>1998</td>
<td>C++</td>
<td>-</td>
</tr>
<tr>
<td>DVSQDN</td>
<td>Davidson</td>
<td>1.0</td>
<td>1995</td>
<td>F77</td>
<td>-</td>
</tr>
<tr>
<td>LANCZOS</td>
<td>Lanczos [N]</td>
<td>-</td>
<td>1992</td>
<td>F77</td>
<td>-</td>
</tr>
<tr>
<td>LANZ</td>
<td>Lanczos [P]</td>
<td>1.0</td>
<td>1991</td>
<td>F77</td>
<td>-</td>
</tr>
<tr>
<td>LASO</td>
<td>Lanczos [S]</td>
<td>2</td>
<td>1983</td>
<td>F77</td>
<td>-</td>
</tr>
<tr>
<td>LOPSNI</td>
<td>Subspace Iteration</td>
<td>1</td>
<td>1981</td>
<td>F77</td>
<td>-</td>
</tr>
<tr>
<td>JDQRI/JDQZ</td>
<td>Jacobi-Davidson</td>
<td>-</td>
<td>1998</td>
<td>F77/Matlab</td>
<td>-</td>
</tr>
<tr>
<td>NA18</td>
<td>Block Davidson</td>
<td>-</td>
<td>1999</td>
<td>F77</td>
<td>-</td>
</tr>
<tr>
<td>NAPACK</td>
<td>Power, Lanczos [N]</td>
<td>-</td>
<td>1987</td>
<td>F77</td>
<td>-</td>
</tr>
<tr>
<td>QMRPACK</td>
<td>Nonsymmetric Lanczos (lookahead)</td>
<td>-</td>
<td>1996</td>
<td>F77</td>
<td>-</td>
</tr>
<tr>
<td>SRRIT</td>
<td>Subspace Iteration</td>
<td>1</td>
<td>1997</td>
<td>F77</td>
<td>-</td>
</tr>
<tr>
<td>SVDPACK</td>
<td>SVD via Lanczos [P], Ritzit &amp; Trace Minimization</td>
<td>1992</td>
<td></td>
<td>C/F77</td>
<td>-</td>
</tr>
<tr>
<td>Underwood</td>
<td>Block Lanczos [F]</td>
<td>1975</td>
<td></td>
<td>F77</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 2.6: Software packages for eigenvalue solutions
2.3.2 SLEPc components

SLEPc stands for Scalable Library for Eigenvalue Problem Computations and is based on the PETSc package (refer to section 2.2). SLEPc can be considered an extension of PETSc which makes use of certain tools and data structures available in PETSc to provide efficient solution for eigenvalue problems.

![Figure 2.7: Components of SLEPc and PETSc](image)

Figure 2.7 [12] provides all the objects provided by PETSc and SLEPc. The details on PETSc structure is already explained in the previous section (refer to section 2.2). The components of SLEPc as shown in the figure utilizes PETSc functionalities and structure to provide advanced solvers and spectral transformations. For this research we analyse the eigensolver layer to provide customized eigenvalue solutions.
2.3.3 SLEPc eigensolver routines

As a first step towards this research, we studied the features [29] and the implementation specifics of the solvers available in SLEPc. This study helped us in obtaining the input characteristics that can impact the convergence of the solvers. The following section elaborates the properties of these eigenvalue solvers.

2.3.3.1 Power, Inverse, Rayleigh Quotient iteration

The three single vector iterations provided by SLEPc are the Power iteration, Inverse iteration and Rayleigh Quotient iteration (RQI). The Power iteration finds the eigenvalue with largest magnitude only. The convergence of Power method is linear and depends on the ratio of the first two eigenvalues. As a result, shifts need to be applied for some cases. If the matrix and eigenvectors are real and the eigenvalues are complex this solver does not converge due to the lack of unique dominant eigenvalue. For such cases, SLEPc recommends using other solvers. Inverse and RQI iterations additionally requires solving a linear system of equations.

2.3.3.2 Subspace

SLEPc recommends Subspace iteration over the previous solvers for cases with complex conjugate eigenvalue pairs when working in real arithmetic [17]. This method, like power method, gives the largest magnitude eigenvalue and supports all types of matrices. This implementation is robust since it is based on Schur decomposition and also implements deflation. SLEPc does not provide many options to change default parameters for this method, for example setting tolerance to detect a group of eigenvalues [18].

2.3.3.3 Arnoldi

This eigensolver works for any kind of matrix and finds eigenvalues in any part of the spectrum (largest, smallest or target) but SLEPc implementation of it cannot find all eigenvalues. SLEPc implements the explicit restart with locking algorithm and claims it to be the most robust SLEPc.
eigensolver [19], although its convergence is slower than that of Krylov-Schur (refer to section 2.3.3.5. For a generalized eigenvalue problem, if matrix $B$ (in $Ax = \lambda Bx$) is singular, this solver may cause large errors.

2.3.3.4 Lanczos

The Lanczos method in SLEPc works only for symmetric matrices, hence only Hermitian and generalized Hermitian eigenvalue problems are supported. This method too finds eigenvalues in any part of the spectrum except the case of finding all eigenvalues. Similar to Arnoldi, the implementation for this method is explicit restart with locking. Different orthogonalizing strategies can be set depending on requirements [20].

2.3.3.5 Krylov-Schur

Krylov-Schur is the default method in SLEPc. This method works for any kind of matrix (Hermitian as well as non-Hermitian). This method can find eigenvalues in any part of the spectrum including all eigenvalues. The restarting scheme implemented in SLEPc is effective and robust and hence provides faster convergence than other solvers. Robustness of this method is equivalent to that of Arnoldi but it is much faster and more effective [21]. The performance results comparing this method with ARPACK are elaborated in the SLEPc technical reports [21].

2.3.3.6 Generalized Davidson

This method works for any kind of matrix and all parts of the spectrum of eigenvalues (except finding all eigenvalues). Davidson solvers give better performance than other solvers in finding interior eigenvalues. An arbitrary preconditioner is applied on the residual vector to get a better approximation in the direction of the desired eigenvector. This method leads to stagnation if the resultant vector is almost collinear with the eigenvector. This method gives good performance for easier problems [23].
2.3.3.7 Jacobi Davidson

This method too works for all eigenvalue problems and gives better performance for interior eigenvalues. Unlike Generalized Davidson, this method depends less on the quality of the preconditioner for convergence. The preconditioner may enhance the quality and efficiency of the solution though. This method seeks to avoid the stagnation mentioned in the previous method [23]. Additionally, Jacobi Davidson works for non-normal matrices as well, hence this method is recommended for complicated problems.

Studying the properties of these eigensolvers helped in setting up the features of the input and output conditions that can impact the results. These features are explained in section 4.1.

2.4 Machine Learning

In this section, we explain some of the machine learning concepts which are used in the research. We describe two popular classification techniques that helped us in creating a model that can intelligently determine the best eigensolver for different user scenarios. Finally we discuss some applications that are used for classification.

2.4.1 Introduction

Machine learning is a branch of computer science that refers to learning from features of data with known output so as to predict any unknown output. There are two different prediction schemes, Supervised and Unsupervised. In the Supervised scheme, the output to be predicted is from a known set of values. In the Unsupervised scheme, the result to be predicted is from an unknown output set. This thesis uses Supervised learning techniques to create a model that can predict the most apt eigenvalue solver depending on the input features which include the properties of the matrix and desired output parameters. Two learning techniques, Decision tree learning and Support vector machine (SVM) scheme are detailed below.
2.4.2 Decision tree

Decision tree induction [16] is one such popular prediction model that uses observations with known results, referred to as the training data set, to form a model for predicting the results for any data. It uses the features or attributes of the data set to form a pattern that can best fit the training data. The pattern created is in the form of a tree known as Decision tree or Classification tree.

Input to a Decision tree model, the training data, is in the form of a table with all but one column as attributes of the data. For example, Table 2.1 shows sample input table format. The last column represents the known classification (class label) of the training data.

<table>
<thead>
<tr>
<th>Age</th>
<th>Credit rating</th>
<th>Student</th>
<th>Buys computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Youth/MiddleAged/Senior</td>
<td>Fair/Excellent</td>
<td>Yes/No</td>
<td>Yes/No</td>
</tr>
</tbody>
</table>

Table 2.1: Sample input structure to Decision tree algorithm

In this example we want to classify whether a person will buy a computer or not based on three attributes of the person: age, credit rating and if the person is a student or not. Note that in this example all attributes are categorical, but numerical attributes are also supported.

![Sample decision tree predicting a computer purchase](image)
A sample decision tree created for training data in the above format is shown in figure 2.8 [16].

Looking at the left most node of the tree, we can interpret (starting from the root node) that if the person’s age is in youth category and the person is not a student, he or she will not buy a computer. In similar manner the entire tree can be converted into a prediction model. As a result, for any future data we can traverse the tree to predict if the person is likely to purchase a computer.

Algorithm

In this section we briefly describe the steps that converts the data from a format shown in table 2.1 into a tree model as shown in figure 2.8.

Let $D$ denote the training dataset containing $n$ points in a $d$ dimensional space (which implies a table with $n$ rows and $d$ columns). The $(d - 1)$ columns in dataset $D$ denotes the attributes of the data and one column in the dataset represents the known class label. Let attributes of each row is represented by $x_i$ and the class label of each row is represented by $y_i$. Let these data points belong to a data space denoted by $R$. Decision tree classifier algorithm recursively partitions the data into a model such that every point in the data space $R$ can be mapped to one of the class labels in the set $y$ [35]. This algorithm uses an axis parallel hyperplane to split the data space $R$ into two half spaces say $R_1$ and $R_2$. These half spaces imply that the dataset $D$ is also partitioned into two respective sets say $D_1$ and $D_2$. Each of these half spaces are now recursively split into further subspaces until most of the points in the subspace denote the same class label [35]. Figure 2.9 [33] depicts this partitioning mechanism for a two dimensional data set. Note that after the first partitioning, the class label $b$ is the sole class on the left partition. The right section is further divided into two sub sections after the second partition. Every section has one dominant class label.

The Decision tree algorithm splits the data space only using an axis parallel hyperplane, which implies that the hyperplane must be parallel to one of the dimensions of the dataset. In other words, every partition in the created hierarchical model must be based on a single attribute of the dataset. Thus, a major part of this recursive algorithm for partitioning is attribute selection
Attribute selection measure [16] of an attribute is a numeric value which depicts how much that attribute contributes to the class label with respect to all the other attributes in the data set. The higher this value, the higher is the contribution of that attribute. Hence the attribute with high score is chosen to partition the data. Various heuristic algorithms are available to calculate the attribute selection measure for every attribute in the dataset. Some of the popular algorithms to calculate the attribute selection measure includes information gain, gain ratio and gini index [16].

The result of this partitioning is a hierarchical model in the form of a (binary) tree, where every node represents an attribute that caused the partition, every branch from the node denotes the subspace with respect to the chosen attribute value and the leaf node leads to the predicted class label.

2.4.3 Support vector machine

Support vector machine (SVM) algorithms, like Decision tree induction, are Supervised learning algorithms to create a model which classifies training data into class labels. For any new data points, this model can be used to predict its class label.

One of the main differences between Decision tree induction and Support vector machine algorithm is that Decision tree inherently classifies the data into multiple attributes, where as the
basic Support vector machine algorithm classifies the data into two sub classes only. Hence, in order to predict multiple attributes using SVM, the algorithm needs to be run one attribute at a time.

Support vector machine algorithms are considered more advanced because, unlike Decision trees, it can be used to model non-linear boundaries [33] as well. SVMs use linear models to implement non linear boundaries by transforming the input using a non linear mapping.

Algorithm

Support vector machine algorithm is based on forming a maximum margin hyperplane [33] which implies finding a optimal hyperplane that maximizes the margin or gap between the classes [35]. To explain the concept of maximum margin hyperplane, consider figure 2.10. The top left graph shows a two dimensional data with two classes a and b.

![Figure 2.10: SVM maximum margin hyperplane](image)

Figure 2.10: SVM maximum margin hyperplane
Note that margin 1 and margin 2 created to separate the two classes have only a small gap between them. Margin 3 on the other hand, has a large gap between the classes. Infact, margin 3 is the largest separation that can be achieved between these two classes. Support vector machine algorithm optimally finds this separation, known as maximum margin hyperplane. The data points that are closest to the maximum margin hyperplane are known as support vectors [33]. These support vectors uniquely defines this hyperplane.

To find the maximum margin hyperplane, the SVM algorithm converts the problem into a constrained (convex) quadratic optimization problem [16]. Various applications are available which can be used to obtain SVM model from the training data.

The next section briefly states the applications that implement different classification techniques.

2.4.4 Machine learning applications

There are various tools available to obtain the Decision tree and Support vector machine models using training data.

MATLAB [27] is one such application popular amongst programmers. Both Decision tree induction and Support vector machines are implemented in MATLAB. Though for SVM, only the base case is available, which implies classification to two sub classes at a time. Open-source multi class SVM implementation is available online [4].

Orange[8] is an open source data mining tool for learning operations such as classification, evaluation and prediction. Unlike MATLAB, it provides an easy to use interface to load and view data. Users with no programming knowledge can also use this application easily. Orange supports Decision tree classification as well as multiple attributes SVM algorithms.

These two applications were used extensively to classify the training data for this research.
Chapter 3

Research Approach

The aim of this research is to provide the best solutions to user-specified sparse eigenvalue problems through the Lighthouse interface. In this case, the solution does not just mean a favorable eigensolver selection but also the complete code that can be downloaded by the user and used for their applications. To achieve this goal we divide this problem into three sub problems: finding the right questions to be asked to the user, finding the right SLEPc solver for any user-specified problem and adding to the Lighthouse interface to accommodate sparse eigensolvers in SLEPc. These tasks are explained in detail in the following sections.

3.1 Finding the right questions

One of the foremost tasks is to obtain the list of questions that needs to be asked of the user to help in understanding the exact problem. These questions must be comprehensive enough such that they lead to a narrowed list of solutions, yet they must be concise that the user does not get impatient and abandon the entire process.

In our case, questions are nothing but the features or attributes of the users problem domain. These attributes include properties of the input matrix and desired output parameters that can impact either the eigensolver selection or the code to be generated. To find these questions we performed two different kinds of analysis. First, we studied the properties of the iterative sparse eigensolvers available in SLEPc. We made note of the input matrix characteristics that each of these solvers support and the solvers that are more favorable to certain result parameters. Second
we analysed the SLEPc manual in detail to find the parameters that need to be set in the code to reach the desired solution. The entire set of features obtained after these two analysis are described in section 4.1.

3.2 Finding the right SLEPc solver

There are variety of SLEPc solvers for different kinds of eigenvalue problems depending on the type of matrix and the kind of eigenvalue solution user seeks. This task involves narrowing down to the most apt eigenvalue solver depending on the answers to questions posed in the previous section.

To find the right solver first we experimented with a number of sample matrices with an attempt to cover a range of values for the features selected. We noted certain resultant parameters for each of the eigensolvers supported by SLEPc. Depending on these results, we algorithmically determined the solver(s) that works the best for every tested case. We then used machine learning techniques to make intelligent decisions about the best known solvers for any untested case. These tasks are explained in detail in the sections 4.2 to 4.4.

3.3 Integration with Lighthouse

The final task is to implement the code and plug it into the Lighthouse interface. This code should support all the selections made by the user for the features and provide an appropriate solver. Integration to Lighthouse also includes adding the questions pertaining to the sparse eigensolvers to the Lighthouse interface and delivering an appropriate solution after user selection.

A MySQL database is constructed using Django constituting the features selected in the first task (refer to section 3.1) and the corresponding best eigensolvers. The Lighthouse interface supports guided search, advanced search and keyword search for sparse eigensolvers. These steps are explained in section 4.7.

The execution of the above three steps is explained in detail in the next section.
Chapter 4

SLEPc Eigensolver Integration

This chapter details the steps involved in obtaining and integrating the best eigensolver routines with Lighthouse. Analysis of eigensolvers available in SLEPc (section 2.3) helped in obtaining the properties of the eigenvalue problems or the sought eigenvalue solutions that could have an effect on the most favorable solver selection. The next section explains the experiments conducted with different eigenvalue problem spaces and available SLEPc solvers. We studied and analysed the characteristics of the results obtained from these experiments algorithmically and narrowed it down to the solvers that works best for each of the experimented problem spaces.

We used these narrowed results as training data to machine learning algorithms. Decision tree learning is best suited to this data specifically because it can easily model classifiers for training data with multiple target classification. We also tried Support vector machine (SVM) for classification and compared the results with Decision tree induction.

We used MATLAB and ORANGE for classification analysis and validated the predicted results using error checking functionalities to ensure low mispredict ratio. Finally we integrated the results with Lighthouse by adding the decision tree information into the Lighthouse database. Depending on user selection the appropriate solvers are fetched from the database for code generation.

4.1 The Feature Set

We studied the features of the input matrix and the desired output parameter for two reasons. First, to obtain the series of questions that need to be asked from the user so that we can provide
the solution they desire. Second, to understand which parameters may influence the outcome of most favourable SLEPc eigensolver so that we can provide the best possible solvers to the user. This section is a compilation of these features with explanation on possible effect on results.

**Size of matrix**

Some eigenvalue solvers may work better for larger matrices while others may work better for smaller matrices.

**Complex or Real**

If the matrix is complex, the SLEPc configuration is modified to support complex numbers. Hence, it may have an effect on the convergence.

**Number of processors**

Since SLEPc supports parallel processors, the number of processors that are used to find the solution may influence the results. This depends on how MPI parallelism is implemented in SLEPc.

**Problem type**

Problem type refers to the matrix categories Hermitian or non-Hermitian in the experiments conducted. Some SLEPc eigensolvers work exclusively for Hermitian matrices, hence depending on the problem type the output or convergence may vary.

**Portion of spectrum**

Portion of spectrum refers to where the sought eigenvalue resides in the entire eigenvalue spectrum. It can range in largest magnitude, smallest magnitude, target magnitude, largest real, smallest real, target real, largest imaginary, smallest imaginary, target imaginary, all eigenvalues. Not all SLEPc eigensolvers support all the mentioned spectrum, hence the effect on the results is noted in the experiments

**Number of eigenvalues**

Some eigensolvers may be fast in finding the first few eigenvalues while others may be better when eigenvalues are needed in large numbers.

**Tolerance**
There is a possibility that, for our problem space, some eigensolver converges to eigenvalue with low residual tolerance much faster than others. To capture that, various tolerance ranges are tested.

**Binary**

Binary matrices have the specific characteristic that every non zero value is the same. This may modify the convergence of eigenvalues.

The following properties of the results are obtained to decide on the most favourable solver.

**Number of converged eigenvalues**

The resultant number of converged eigenvalues has a lower limit, it must be at least the number desired in the defined problem space.

**Residual**

Similar to the previous property, the residual must be less than or equal to the tolerance value given as input.

**Time taken**

This is the most important property when looking for the best solver. Out of all the solvers that meet the criteria desired by the user, we want to select the one that takes least amount of time to converge.

### 4.2 Experimental setup

This section describes the steps taken to conduct experiments so as to obtain effect on the results of eigenvalue problem for different input features. First we detail on the machine architecture and software details that were used to run these experiments. Next, three tasks that lead to experimental results are explained, analysis of SLEPc implementation, SLEPc code for eigensolvers and running the experiments.
4.2.1 Test bed

We first detail on the test bed set for the experiments. All the experiments for this research were conducted on Janus supercomputer. The table 4.1 gives some basic information. More information about Janus supercomputer can be found online [1].

<table>
<thead>
<tr>
<th>OS</th>
<th>Redhat 6 Enterprise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute nodes</td>
<td>1368</td>
</tr>
<tr>
<td>Total cores</td>
<td>16416</td>
</tr>
<tr>
<td>RAM per core</td>
<td>2GB</td>
</tr>
<tr>
<td>RAM per node</td>
<td>24GB</td>
</tr>
<tr>
<td>Lustre filesystem storage</td>
<td>800TB</td>
</tr>
</tbody>
</table>

Table 4.1: Janus supercomputer details

The gcc compiler version used was 4.4.7. SLEPc version 3.3 was used for all experimentation.

4.2.2 Analysis of SLEPc implementation

Different matrix properties and desired output parameters are already discussed in the previous section. In order to run the experiments, the next step was to find where to set these matrix properties and output parameters. If the parameters/properties can be set through the command line as well as in the C code, the command line argument is given preference.

The easiest way to set values for SLEPc eigensolvers is to set them on the command line while executing the eigensolver code. A respective command line key is passed for each property or the desired output parameter. As we know that SLEPc is built on top of PETSc libraries, there are certain input parameters that need to be set while configuring PETSc. This is a little harder than the simple command line argument case. In order to do so, the ‘./configure’ command needs to
be called along with the desired command line key arguments in the `PETSC_DIR`. The entire setup along with the SLEPc configuration needs to be completed to change such parameters.

If the matrix is complex, this value needs to be set at the PETSC configuration level. This value is set to real by default. The command line key to change it to complex is `-with-scalar-type=complex`. Even to modify the single precision support, the value needs to be set at the PETSC configuration level. By default the precision is set to double. The command line argument to set it to single is `-with-precision=single`.

Problem type refers to the eigenvalue problem type which is Hermitian or non-Hermitian. Currently we are not considering the generalized case. Problem type of the input matrix can be set at command line while executing the code. The command line key for Hermitian is `-eps_hermitian` and that of non-Hermitian is `-eps_non_hermitian`.

Portion of spectrum refers to (as explained in previous section) which portion of the eigenvalue spectrum we desire, such as, largest magnitude, smallest real, etc. Like most other properties, the portion of spectrum user desires can be set on the command line while executing the code. The parameters used for coding different eigenvalues are `-eps_largest_magnitude`, `-eps_smallest_magnitude`, `-eps_largest_real`, `-eps_smallest_real`, `-eps_largest_imaginary`, `-eps_smallest_imaginary`, `-eps_target_magnitude`, `-eps_target_real`, `-eps_target_imaginary` and `-eps_all`.

The number of eigenvalues desired, can also be set as a command line key while executing the code. The number of eigenvalues sought needs to be passed after `-nev`, for example for two eigenvalues, we use `-nev 2`.

The number of processors to run the code in parallel can also be set as a command line key while executing the code. While running the code the executable needs to be prefixed with `mpirun -np` followed by the number of processors. For example, to run in 8 processors, we use

```
mpirun -np 8 .eigenvalue_solver
```

The tolerance parameter is also set on the command line while executing the code. It is set by using the key `-eps_tol` followed by the tolerance value. For example, to set the tolerance as 0.0001, we use `-eps_tol 0.0001`.
The eigensolver that needs to be used for solving the eigenvalue problem can be set on the command line while running the executable. KrylovSchur is set by default. The others can be set using the key -eps_type on the command line followed by the method name. The method names are power, subspace, arnoldi, lanczos, gd, jd, and rqcg.

4.2.3 SLEPc code for eigensolver

The SLEPc code implementation consists of two parts, loading the matrix and solving the eigenvalue problem.

Loading the matrix (matrix_load.c)

Input matrices in the matrix market file format are supported. In this format, every line in the file gives the row and column information followed by the value of the element in that row and column. We read the value into a PetscScalar object. For real matrices, we read only one value and set that value to the PetscScalar object. For complex matrices, we read two values, the first value is set as the real part of the PetscScalar object and second value is set as the imaginary part. For a binary matrix, only row and column numbers are given, the value is assumed to be one.

Parallel implementation

To run eigenvalue solvers in multiple processors, the loading of the matrix still has to be sequential. To make it easier, we read the file in matrix market format sequentially and save it in Petsc Binary format using PetscBinaryViewerOpen(). While executing the eigensolver, we read the matrix in parallel using the newly created binary file.

Solving for eigenvalue (eigenvalue_solver.c, eigenvalue_parallel.c)

After reading the binary file in parallel, the next step is solving the eigenproblem. We implemented the basic eigensolver in C making no changes to the default settings. This is because any matrix property or output specific parameter is set as a command line argument while executing the code. We created script files to modify the command line parameters, a detailed explanation is provided in the next section.

For ease of saving results to Excel, every eigenvalue solver run writes the output in a single
4.2.4 Script file generation for experiments

For experimenting with different permutations of input parameters we created two script files.

The `experiments.sh` file consists of different permutations of experiments that are run for different sample matrices. The parameters that were experimented with are number of processors, portion of spectrum, number of eigenvalues sought, tolerance value for residual and the eigensolver method. This script file is written such that these experimental parameters can be easily modified for checking different results. Also note that these parameters are independent of the input matrix, hence the parameter values can be modified according to experimentation needs for all sample matrices. The inputs to this file are the name of the sample matrix file, the type of matrix (Hermitian or non-Hermitian) and if values are real or complex.

The `build_all.sh` script file is the main file that runs the experiments. The first step is to check if PETSc and SLEPc are set up properly. Any possible error in the setup is avoided by checking the environment variable associated with both PETSc and SLEPc. If the variables `PETSC_DIR`, `SLEPC_DIR` and `PETSC_ARCH` are not set, an error is reported and the file exits. If PETSc and SLEPc are set up in the machine, the configuration is modified depending on whether the sample matrix is real or complex. This file consists of the sample matrix filenames which are the input to all the experiments. Additionally, certain parameters such as whether the matrix is complex or real and whether the matrix is Hermitian or not are also specified in this file depending on the input. These matrix property information are sent to `experiments.sh` for appropriate use. To test a new matrix, the filename needs to be added to this file along with the matrix information on the type (Hermitian or not) and support for complex numbers.
4.3 Running the experiments

To run the experiments, we first obtained matrices from Matrix Market [3] and the Florida Sparse Matrix Collection[2] that cover the problem domain of interest.

4.3.1 Data

The Table 4.2 lists the sparse matrix files with respective properties. These files are in Matrix market format.

A varied range of data files was obtained for experimentation with size ranging from 112x112 being smallest to 262111x262111 being the largest. Different combinations of real and complex and Hermitian and non-Hermitian matrices were tested. We also added a few binary matrices.

4.3.2 Experiments

For each of the files in table 4.2, the results were noted. Table 4.3 shows the range of input parameters tested. The results were obtained for most permutations of these inputs; the resultant data set consists of more than 29,000 data points.

4.4 Training and Prediction

By running these experiments, we obtained the output characteristics of different matrices under different conditions. Output characteristics we are interested in are number of converged eigenpairs, time taken and residual. In order to narrow down to a solver that works best for certain input condition, the first step is to find the best solvers for an input set. Once we have a unique set of solvers for every unique input, we use machine learning to make intelligent decisions on the most favourable solvers.
<table>
<thead>
<tr>
<th>Filename</th>
<th>Real or Complex</th>
<th>Hermitian or Non-Hermitian</th>
<th>Binary</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcsstk03.mtx</td>
<td>Real</td>
<td>Hermitian</td>
<td>No</td>
<td>112 x 112</td>
</tr>
<tr>
<td>bwf398b.mtx</td>
<td>Real</td>
<td>Non-Hermitian</td>
<td>No</td>
<td>398 x 398</td>
</tr>
<tr>
<td>plat1919.mtx</td>
<td>Real</td>
<td>Hermitian</td>
<td>No</td>
<td>1919 x 1919</td>
</tr>
<tr>
<td>mhd4800b.mtx</td>
<td>Real</td>
<td>Hermitian</td>
<td>No</td>
<td>4800 x 4800</td>
</tr>
<tr>
<td>af23560.mtx</td>
<td>Real</td>
<td>Non-Hermitian</td>
<td>No</td>
<td>23560 x 23560</td>
</tr>
<tr>
<td>bcsstm39.mtx</td>
<td>Real</td>
<td>Hermitian</td>
<td>No</td>
<td>46772 x 46772</td>
</tr>
<tr>
<td>shipsec1.mtx</td>
<td>Real</td>
<td>Hermitian</td>
<td>No</td>
<td>140874 x 140874</td>
</tr>
<tr>
<td>stomach.mtx</td>
<td>Real</td>
<td>Non-Hermitian</td>
<td>No</td>
<td>213360 x 213360</td>
</tr>
<tr>
<td>mhd1280a.mtx</td>
<td>Complex</td>
<td>Non-Hermitian</td>
<td>No</td>
<td>1280 x 1280</td>
</tr>
<tr>
<td>mhd1280b.mtx</td>
<td>Complex</td>
<td>Hermitian</td>
<td>No</td>
<td>1280 x 1280</td>
</tr>
<tr>
<td>kim1.mtx</td>
<td>Complex</td>
<td>Non-Hermitian</td>
<td>No</td>
<td>38415 x 38415</td>
</tr>
<tr>
<td>amazon0302.mtx</td>
<td>Real</td>
<td>Non-Hermitian</td>
<td>Yes</td>
<td>262111 x 262111</td>
</tr>
<tr>
<td>as-735.mtx</td>
<td>Real</td>
<td>Hermitian</td>
<td>Yes</td>
<td>7716 x 7716</td>
</tr>
<tr>
<td>email-Enron.mtx</td>
<td>Real</td>
<td>Hermitian</td>
<td>Yes</td>
<td>36692 x 36692</td>
</tr>
</tbody>
</table>

Table 4.2: Sparse matrix files used in experiments
<table>
<thead>
<tr>
<th>Property tested</th>
<th>Range of sample tested</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix order</td>
<td>112 x 112 to 262111 x 262111</td>
</tr>
<tr>
<td>Matrix type</td>
<td>Real, Complex</td>
</tr>
<tr>
<td>Matrix data</td>
<td>Binary, Double</td>
</tr>
<tr>
<td>Matrix characteristics</td>
<td>Hermitian, Non-Hermitian</td>
</tr>
<tr>
<td>Number of eigenvalues</td>
<td>1, 2, 5, 10</td>
</tr>
<tr>
<td>Portion of spectrum</td>
<td>Largest magnitude, Smallest magnitude, Largest real, Smallest real, Largest imaginary, Smallest imaginary</td>
</tr>
<tr>
<td>Tolerance</td>
<td>1.00E-04, 1.00E-08, 1.00E-10</td>
</tr>
<tr>
<td>Number of processors</td>
<td>1, 2, 4, 8, 12, 24, 48, 96, 192</td>
</tr>
</tbody>
</table>

Table 4.3: Input parameters tested
4.4.1 Finding optimum solver (Training data setup)

For exactly the same set of inputs, we collect the results for different eigenvalue solvers. For example, consider a sample test case for the sparse matrix file shipsec1.mtx in table 4.4.

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>Complex or Real</th>
<th>Number of processors</th>
<th>Problem type</th>
<th>Portion of spectrum</th>
<th>Number of eigenvalues</th>
<th>Tolerance</th>
<th>Binary</th>
<th>Eigensolvers tested</th>
</tr>
</thead>
<tbody>
<tr>
<td>140874</td>
<td>Real</td>
<td>24</td>
<td>Hermitian</td>
<td>Largest magnitude</td>
<td>2</td>
<td>0.00000001</td>
<td>No</td>
<td>power, subspace, arnoldi, lanczos, krylovschur, gd, jd</td>
</tr>
</tbody>
</table>

Table 4.4: Sample test case from sparse matrix file shipsec1.mtx

Note from the table 4.4 that for one set of input, the resultant characteristics are obtained for each of the applicable SLEPc solvers. Table 4.5 shows sample result data obtained by running these experiments with each applicable solver.

<table>
<thead>
<tr>
<th>Eigensolver</th>
<th>Number of converged eigenpairs</th>
<th>Residuals</th>
<th>Time taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>power</td>
<td>2</td>
<td>5.46E-09, 5.99E-09</td>
<td>0.44</td>
</tr>
<tr>
<td>subspace</td>
<td>2</td>
<td>2.06E-09, 1.44E-09</td>
<td>2.7</td>
</tr>
<tr>
<td>arnoldi</td>
<td>3</td>
<td>3.23E-09, 1.86E-09, 9.01E-09</td>
<td>0.2</td>
</tr>
<tr>
<td>lanczos</td>
<td>3</td>
<td>3.23E-09, 1.86E-09, 9.01E-09</td>
<td>0.19</td>
</tr>
<tr>
<td>krylovschur</td>
<td>3</td>
<td>7.39E-10, 1.86E-09, 3.37E-09</td>
<td>0.21</td>
</tr>
<tr>
<td>gd</td>
<td>2</td>
<td>5.51E-09, 6.97E-09</td>
<td>0.44</td>
</tr>
<tr>
<td>jd</td>
<td>2</td>
<td>2.29E-10, 6.54E-09</td>
<td>0.39</td>
</tr>
</tbody>
</table>

Table 4.5: Sample results from sparse matrix file shipsec1.mtx for input case in table 4.4

Once we have this data, the next part of the experimental analysis deals with finding the
solvers that gives the most optimum results. We wrote MATLAB code (Appendix B) to find the best solver depending on the importance of each of the output characteristics. This analysis can be split into two steps, Elimination and Selection.

In the Elimination step, we first remove the solvers with resultant characteristics completely outside the desired value. The two resultant properties expected of the output are the number of specified eigenvalues and the tolerance for the residual. The number of eigenvalues obtained with the residual within the specified tolerance are counted. If this count is less than the number of eigenvalues desired, we remove the solver from the list. After this elimination step, we have the eligible solvers that converge to desired results.

The next step is the selection of the most appropriate solvers. The solver which takes the least amount of time is selected to be the best fit for the given conditions. Additionally, any solver which takes a total time within 110% of the best fit solver is also included in the selected list.

For every input permutation tested, we follow the method detailed above to obtain the resultant table (a subset of the original dataset) which identifies the best solvers for each unique input set.

For example, consider table 4.5, each of the solvers converged to the desired residual tolerance and to the desired total number is eigenvalues. Hence, none of the solvers are removed for the elimination step. For the selection step, we note that lanczos takes the least amount of time, 0.19 seconds, therefore we select it as the best fit. Additionally, any solver within 110% of the time taken by the best fit, (0.209 seconds for this case) is also selected. Note that arnoldi method makes it to the list with this process. Hence, for the input case given in table 4.4, lanczos and arnoldi are the favorable solvers.

Handling cases with no convergence

After experimenting with very large upper limits, we set the maximum iteration limit to 1000 iterations for all eigensolvers except the power method. The time to run an iteration of the power method is much less than the time for other solvers, hence we set the upper limit for it to 5000. The experiments established that most solutions converge at iteration counts much less than these
values. If the solution does not converge within the specified limits, it is labelled as non-convergent.

In the case where none of the eigensolvers converges to the desired number of eigenvalues or the desired tolerance, there is still a possibility that a solver converges but not to the exact expected results. If so, a solver that shows some signs of convergence, even though not to the desired values, is retained for the training data. Hence, along with ‘No Convergence’, the partially converged solvers are added to the resultant table.

4.4.2 Intelligent solver selection (Prediction)

From the previous step we have this reduced set of data which gives the optimum solvers for each case. After some preprocessing to match our needs to predict multiple solvers, we feed this reduced set of best solvers with respective input criteria to MATLAB and ORANGE prediction solvers. We used Decision tree induction (refer to section 2.4) to build a classification tree which helped us in creating a model that indicates the most favourable solvers for an input condition. We created another classification model using Support vector machine (SVM) algorithm also. The following section describes these schemes in detail.

4.4.2.1 Preprocessing data for classification

In order to accommodate multiple solver classification for the obtained training data, we added a feature ‘Rank’ to our data table. For each of the best solvers obtained from the previous step we assign a rank to it based on respective time taken to converge. The best fit solver is given the rank one and other solvers are given incremental ranks with increasing time taken. For the example mentioned for the input case in table 4.4, we selected lanczos and arnoldi, hence based on the time taken we assign them the rank as shown in table 4.6.

As shown in the table 4.6, the training data now has nine features including rank. These features are used to create our classification model as explained in the next few sections.
Table 4.6: Sample test case with rank from sparse matrix file shipsec1.mtx

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>Complex or Real</th>
<th>Number of processors</th>
<th>Problem type</th>
<th>Portion of spectrum</th>
<th>Number of eigenvalues</th>
<th>Tolerance</th>
<th>Binary</th>
<th>Rank</th>
<th>Selected eigen solvers</th>
</tr>
</thead>
<tbody>
<tr>
<td>140874</td>
<td>Real</td>
<td>24</td>
<td>Hermitian</td>
<td>Largest magnitude</td>
<td>2</td>
<td>1E-08</td>
<td>No</td>
<td>1</td>
<td>lanczos (0.19 sec)</td>
</tr>
<tr>
<td>140874</td>
<td>Real</td>
<td>24</td>
<td>Hermitian</td>
<td>Largest magnitude</td>
<td>2</td>
<td>1E-08</td>
<td>No</td>
<td>2</td>
<td>arnoldi (0.20 sec)</td>
</tr>
</tbody>
</table>

### 4.4.2.2 MATLAB Decision tree

For our case, we want the data to be classified into one of the eigensolvers available in SLEPc. This classification was done using `ClassificationTree.fit` function [27] in MATLAB. Information regarding which of the given features are categories (and not numeric) was passed into the function. Figure 4.1 shows part of the pruned decision tree created in MATLAB for real matrices. The upper right corner shows the first partition created by the algorithm based on the problem type. Depending on the given problem this tree can be traversed to reach the most favorable solver at the leaf of the tree.

For example, with respect to figure 4.1, if the user wants to find smallest real eigenvalue for a non-Hermitian matrix, we traverse right on the `ProbType` node for non-Hermitian case and right on the next node for smallest real spectrum. In this manner we obtain krylovshur as the best eigensolver for the given case.

### 4.4.2.3 Orange Decision tree

We also used the application Orange [8] to classify and view the decision tree. In particular, Orange has good visualization options that make viewing large trees much easier than with other applications like MATLAB or Weka [15]. Orange also consists of a number of tools to plot and evaluate the original data and the results (generated after running the classification algorithms).
Figure 4.1: Part of the generated classification tree in MATLAB
A portion of the classification tree generated from Orange is shown in Fig. 4.2. Note that every node, in addition to the splitting property also shows the eigensolver of highest probability. Under the predicted eigensolver, it gives the total chance, in percentage, that can result in the predicted solver. The pie chart next to every node illustrates the numerical proportion of each eigensolver.

Using these applications, we obtained a classifier in the form of a binary tree, with every node representing a feature such as matrix size, matrix type and desired eigenvalue spectrum. The decision to follow the left branch or the right branch of a node is made depending on the value of the feature at that node. The leaf node gives the suggested eigensolver for the path followed. The tree also has the information about the matrix properties and output characteristics which may not converge to a solution for any of the eigensolvers available in SLEPc. Such leaf nodes have the
value set to ‘No Convergence’. Even though rank is included as one of the features of the tree, any leaf node that satisfies the user provided matrix criteria, irrespective of the rank, are included as the desired eigensolver.

4.4.2.4 Orange SVM classification

We also tried to classify the obtained training data using support vector machine classification provided in Orange. Setting up the classifier is only a matter of dragging and dropping the different steps associated with classification on to the Orange canvas and making respective connections between these steps. Various algorithms, like linear, polynomial, RBF, sigmoid, are available for SVM classification. We used a special option provided by Orange for SVM classification known as automatic parameter search algorithm. This method uses learning techniques to find the optimal parameters fit for the given data. The algorithm selected using this search on our data was RBF or radial basis functions [14].

The performance scores with respect to each of these classification techniques are detailed in the next section.

4.5 Result validation

In this section we explain various methods to validate the results obtained using machine learning techniques. Finding the cross-validation error is one of the popular validation techniques used to verify classification algorithms like Decision tree and SVM. A basic cross-validation method divides the data into two segments. The method then uses one segment as training data for learning algorithm and the other segment to validate the model created using the first segment [28]. The two segments are repeatedly selected from the set such that the training data and validation data cross over multiple times.

A k-fold cross validation [28] means dividing the data into k segments and using k-1 segments for the learning algorithm and one segment for validation. This procedure is repeated for k iterations, keeping a different segment for validation each time. Figure 4.3 briefly depicts the division
into k segments and the cross validation technique.

![Cross-validation technique](image.png)

Figure 4.3: Cross-validation technique

The next section details k-fold validation results for our classification schemes. We used this scheme as a preliminary step for validation. Lighthouse application will suggest multiple solvers that are most apt for each input case, hence we wrote an algorithm in MATLAB to obtain the actual mis-predict ratio irrespective of the rank. This technique was only used for the Decision tree created using MATLAB, the details of the results are explained in the section below.

### 4.5.1 Decision tree classification in MATLAB

MATLAB cross-validation results were checked using the MATLAB `cvLoss()` function. Using `cvLoss()`, we obtained the classification error through ten-fold cross-validation for the generated decision tree. The results of `cvLoss()` function are shown in the Table 4.7.

<table>
<thead>
<tr>
<th>$E$</th>
<th>$SE$</th>
<th>$Nleaf$</th>
<th>$BestLevel$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1450</td>
<td>0.0039</td>
<td>395</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.7: Decision tree cross validation results from MATLAB

The cross-validation classification error (loss), $E$, for the generated decision tree was obtained to be 0.1450 which implies a 14.5 percent error in prediction. The standard error of $E$, shown in the
second column as $SE$, was obtained as 0.39 percent. The \texttt{cvLoss()} also gives information about the total number of leaf nodes which is 395 in our case.

This scheme to validate multiple eigenvalue solvers is too stringent as it gives more importance to the rank. Lighthouse will provide all the best solvers for given input conditions irrespective of the rank hence we used a different scheme for validating data specifically for Lighthouse results. We used the training data as input to the generated tree and predicted the results for each feature. For a given input condition, the best solvers subset was obtained. This best solver subset consists of the solvers for each rank in the training data for the specified input condition. If this best solver subset is same as the best solvers in learning set for same input conditions, irrespective of the order, we consider the predictions correct. For example, if for an input condition, rank set as one predicts the best solver as arnoldi and rank set as two predicts the best solver krylovschur, then we obtain this subset as \{\textit{arnoldi}, \textit{krylovschur}\}. Now if the original data for each of the ranks was \{\textit{krylovschur}, \textit{arnoldi}\} respectively, Lighthouse will predict this set correctly, hence we consider this a correct prediction. This method of validation gave a mispredict ratio of 7.99 percent, which implies 92.1% of the data was correctly predicted.

### 4.5.2 Decision tree classification in Orange

As mentioned before, the classification tree generated through Orange is different from that created in MATLAB. Orange also provides various options to evaluate the results. To compare the evaluation results of Orange with the decision tree, we used 10-fold cross-validation for the Orange results as well. Orange provides various other performance scores too. Table 4.8 shows some of these values.

<table>
<thead>
<tr>
<th>CA</th>
<th>Sens</th>
<th>Spec</th>
<th>Prec</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8639</td>
<td>0.8851</td>
<td>0.9083</td>
<td>0.8575</td>
<td>0.8851</td>
</tr>
</tbody>
</table>

Table 4.8: Decision tree cross validation results from Orange
CA implies the classification accuracy, which means the portion of correctly classified samples [8]. CA value of 0.8639 implies 86.39 percent of the results were correctly classified, in other words, 13.61 percent error in prediction. Note that this error value is very close to that obtained for MATLAB classification. Sensitivity, $Sens$, also known as true positive rate [8] is the number of detected positive examples among all the positive examples, which implies for a class, say arnoldi, the number of samples for which the model correctly predicted arnoldi amongst all samples with arnoldi as actual best result. Similarly, specificity, $Spec$, is the ratio of negative examples amongst all negative examples. $Prec$, or precision, is the number of correct class predictions to the total number of class predictions. $Recall$, is the number of correct class predictions to the total number of that class in the training data. These scores can be important to individually verify each class.

The scores $Sens$, $Spec$, $Prec$ and $Recall$ shown in the table are for the class krylovshur.

4.5.3 Support vector machine in Orange

The validation techniques mentioned in the previous section for Orange was also used to obtain the scores for SVM classification. Out of all the algorithm tried for SVM, Table 4.9 shows the best scores obtained.

<table>
<thead>
<tr>
<th>$CA$</th>
<th>$Sens$</th>
<th>$Spec$</th>
<th>$Prec$</th>
<th>$Recall$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8313</td>
<td>0.9078</td>
<td>0.8689</td>
<td>0.8118</td>
<td>0.9078</td>
</tr>
</tbody>
</table>

Table 4.9: SVM cross validation results from Orange

Note that the classification accuracy score is lower than that obtained using decision tree. 83.13 percent accuracy implies an error of 16.87 percent. This is the best score obtained using the automatic parameter search algorithm for SVM in Orange. The algorithm selected using this search was RBF, radial basis functions [14] and the parameters for this function were automatically selected. The performance scores, namely $Sens$, $Spec$, $Prec$ and $Recall$ are shown for the krylovshur eigensolver.
4.6 Result Analysis

As seen in the previous section the decision tree classification scheme works better for the training data, hence we used the results from this classification to integrate with Lighthouse. In this section we discuss a few interesting results obtained. Orange consists of various visualization tools hence plots shown here are generated using Orange. The analysis deduced here is verified with the decision tree results.

4.6.1 Eigensolver vs matrix properties

Plot 4.4 shows a scatter plot of the portion of spectrum tested versus eigensolver. One of the points to note here is that jacobi davidson (jd) is not a preferred solver for any smallest imaginary case. Also note that generalized davidson (gd) is less likely to be selected as the most efficient solver for largest portion of the spectrum.

Figure 4.4: Analyzing eigensolver vs portion of spectrum
Figure 4.5 plots the eigensolver with the binary matrices. Here we note that lanczos solver, which predominantly worked for non-binary matrices, does not work at all for binary matrices, even for Hermitian cases. Similarly, generalized davidson (gd) is also not the most favorable solver for binary matrices.

4.6.2 Spectrum vs Problem type

The plot of portion of spectrum versus problem type is shown in figure 4.6.

First point to note here is that, for finding smallest real eigenvalue for a non-Hermitian matrix the preferred solver is dominantly krylovshur and arnoldi. No other solver gives a better result for these cases. The resultant decision tree also follows the same deductions. Secondly, for Hermitian matrices, the results were observed by setting the input parameter for magnitude value as well as real value. Note that for Hermitian cases, eigenvalues do not have imaginary part. It was observed,
as seen from the plot, that the two spectrum portions (magnitude and real) behave exactly similar. This means that the SLEPc implementation is independent of the magnitude versus real input for Hermitian matrices.

![Figure 4.6: Analyzing portion of spectrum vs problem type](image)

### 4.6.3 Numeric property plots

Figure 4.7 shows the plot of matrix size with residual tolerance. Note the absolute similarity in each of the cases plotted. None of the eigensolvers stand out and the results look very similar. This indicates that these two parameters have very less influence on the decision tree, in other words, the gain in information due to these features is minimal. Hence, these properties are found away from the root of the tree.
A similar result is seen when plotting number of eigenvalues sought versus residual tolerance as shown in plot 4.8. There is no difference in the result with changing parameters and no eigensolver stands out with respect to the properties plotted.

4.7 Integration with Lighthouse

After collecting the required data and verifying the results the next step is to integrate these results with Lighthouse. The goal is to provide access to the users so that the best eigenvalue solution can be generated in a user friendly manner. Updating the Lighthouse user interface is a
Figure 4.8: Analyzing number of eigenvalues vs tolerance
two step process. The first step is constructing the database to upload the obtained results and second step is generating the views to support various search options. The following two subsections describe these tasks in detail.

4.7.1 Database construction

To integrate SLEPc with Lighthouse, we designed and constructed a MySQL database through Django. The database scheme is derived from the information on the decision tree. This database constitutes the attributes from the tree including matrix properties such as matrix type and matrix order as well as the desired output characteristics such as the tolerance and the number of eigenvalues. The following section discusses the design and construction of the database pertaining to SLEPc.

To make the results from this research readily available to the users, two table schemes were constructed.

4.7.1.1 Table for SLEPc routine information

Unlike LAPACK, SLEPc only has a small number of eigensolver routines. Many combinations of input parameters can result in the same SLEPc routine, which means that when traversing the decision tree there are multiple paths that lead to the same solver. Hence we decided to save the solver routine information as a separate table in the database. This table consist of all the distinct eigensolver leaf nodes including the case of no convergence.

4.7.1.2 Table for saving the decision tree model

The resultant decision tree constructed in MATLAB consists of 395 leaf nodes. Converting this model data to a flat table is challenging. Appendix C shows the decision tree generated in text format using MATLAB. A snapshot of the tree is shown below.

1 if Spectrum in {2 4} then node 2 elseif Spectrum in {1 3 5 6} then node 3 else krylovschur
if ProbType=1 then node 4 elseif ProbType=3 then node 5 else NoConvergence

if Spectrum in {1 3} then node 6 elseif Spectrum in {5 6} then node 7 else krylovschur

if Binary=0 then node 8 elseif Binary=1 then node 9 else NoConvergence

if Size<696 then node 402 elseif Size>=696 then node 403 else lanczos

class = NoConvergence
class = lanczos
class = gd
class = jd
class = jd
class = gd
class = krylovschur

Every line of this file represents either a node with two options (note that the third option is never reachable) for example lines 1, 2, 3, 4, 293 or the leaf node with a predicted class, for example lines 294 to 300.

We wrote a python script to read the file and convert it into a flat table using the depth first search algorithm [13] [31]. When the script finds a leaf node, it adds all the encountered property information into a row in the table. Since there are 395 leaves to this tree, we have 395 rows in the generated table.

To reduce the size of the table, a technique extensively used in the Lighthouse database for packages like LAPACK was used for SLEPc as well. The first question splits this depth first search generated tree into two sections. Thus, instead of saving 395 rows as a single table, we split the table into two (or more) tables with respect to the first node. Hence we created one table for first choice traversal (with respect to spectrum) and another table for second choice traversal. The reduced table size helps in faster access of data.
Table data type: Another challenge arises while setting up the table from a decision tree. The property represented by a node is not always a single value but rather may be a range of values. For numeric values, the splitting criteria partitions the tree into a continuous range in floating point, whereas for categorical partitions, the tree splits into a subset of all possible categories. For example, for the path traversed to a leaf node, the number of eigenvalues may be any value between 3.5 and 7, and the spectrum may be any of the values in the set (largest magnitude, largest real, largest imaginary). To accommodate numeric values, instead of saving a single column, we created two columns for such properties, one pertaining to lower limit and the other to the upper limit. For categorical values, we used the Django data type called commaSeparatedInteger to save a list of encountered categories.

Queries: Once the table is set up, the queries to the table are straightforward. Any property selected by the user having numeric values, must pass two criteria, firstly, it should be greater than or equal to the lower limit in the row and secondly, it should be less than the upper limit in the row. For categorical properties, the query must search for ‘IN’ relationship rather than an ‘equal to’ relation.

To handle multiple best solver results, we add to the query all feature criteria except rank. This implies that user makes selection for all features except rank, and the rows that satisfy the user specified feature values are fetched. This may result in multiple rows with multiple solvers differing in rank. All the fetched solvers are displayed to the users irrespective of its rank, thereby providing the best solver subset to the user.

In this manner, a query to the database with a unique selection of these attributes (except rank) will lead to the appropriate eigensolvers predicted by the decision tree.

4.7.2 Lighthouse UI design

This section briefly details the user interface implemented to let user make appropriate choices and generate respective SLEPc code.

Like the LAPACK UI in Lighthouse, the SLEPc UI provides guided search. The questions
in it are established by the decision tree. Each user selection will be a fundamental query to the
database that directs the search toward the most appropriate results. Upon the user’s answering
all the questions, Lighthouse will suggest the best known eigensolvers, and the user may choose to
generate a respective code template in C or Fortran. A part of the current development version of
guided search UI for SLEPc is shown in figure 4.9

The advanced search implementation will contain questions similar to guided search but will deliver

![Figure 4.9: SLEPc guided search UI (under development)](image)

all of the SLEPc eigensolver routines that are compatible with the user-specified problem (irrespec-
tive of the suggestions from the decision tree). The advanced search feature is for experienced users
who are aware of the routines and would like to make their own selections. Additionally, advanced
search will include questions requiring extended eigensolver knowledge.
Chapter 5

Conclusion

We described our methods for easing the most significant task in solving a sparse eigenvalue problem. Our approach includes providing customized SLEPc eigenvalue solutions and generating the respective code through Lighthouse. The intelligent solver suggestions obtained through machine learning techniques offer more accurate and efficient solutions and relieve the users from having to research the documents of every eigensolver routine available. The classification schemes used are verified using validation techniques to ensure accurate predictions.

The automatically generated code will help save time and effort in implementation, thereby improving the productivity of developers and scientists. It will also overcome the barriers caused by unfamiliarity of the programming language and implementation specifics such as parallel programming.

In future, this research can be extended to support additional sparse eigensolver software packages so that the decision tree model is finely tuned to suggest the best solver amongst all the packages in the taxonomy.

We believe that our effort will help in reaching out to a wider user base by providing easy access to computationally challenging sparse eigenvalue solutions, further accelerating scientific development and discoveries.
Bibliography


Appendix A

SLEPc Code - Finds eigenvalue solution

matrix_load.c

```c
/* Program usage: mpiexec ex1 [−help] [all PETSc options] */

static char help[] = "Uploads matrix in binary for parallel use.\n\n";

*/

Main operation: Uploads matrix in petsc binary format for parallel use
Input file format: Matrix market format (matrix and rhs in the same file)
Processor: 1 (sequential)

#include <petsct ime.h>
#include "petscsys.h"
#include <petscmat.h>

#undef _FUNCT_
#define _FUNCT_ "main"

int main(int argc,char **args)
{
    Mat A;
    Vec b;
    char filein[PETSC_MAX_PATHLEN],buf[PETSC_MAX_PATHLEN];
```
PetscInt   i,m,n,nnz,col,row;
PetscErrorCode ierr;
PetscMPIInt   size;
double       re, im;
PetscScalar   val;
FILE*        file;
PetscRandom   r;
char   fileout [PETSC_MAX_PATH_LEN];
PetscBool   isComplex=0;
PetscBool   isBinary=0;
PetscBool   isHermitian=0;

PetscInitialize(&argc,&args,(char *)0,help);

ierr = MPI_Comm_size(PETSC_COMM_WORLD,&size);CHKERRQ(ierr);
if (size > 1) SETERRQ(PETSC_COMM_WORLD,1,"Uniprocessor Example only\n");

/* Read in matrix and RHS */
ierr = PetscOptionsGetString (PETSC_NULL,"−fin",filein ,PETSC_MAX_PATH_LEN,
PETSC_NULL);CHKERRQ(ierr);
ierr = PetscOptionsGetString (PETSC_NULL,"−fout",fileout ,PETSC_MAX_PATH_LEN,
PETSC_NULL);CHKERRQ(ierr);
ierr = PetscOptionsGetBool (PETSC_NULL,"−c",&isComplex ,PETSC_NULL);CHKERRQ(ierr);
ierr = PetscOptionsGetBool (PETSC_NULL,"−b",&isBinary ,PETSC_NULL);CHKERRQ(ierr);
ierr = PetscOptionsGetBool (PETSC_NULL,"−h",&isHermitian ,PETSC_NULL);CHKERRQ(ierr);
ierr = PetscFOpen (PETSC_COMM_SELF, filein ,"r",&file );CHKERRQ(ierr);

/* process header with comments */
do  fgets (buf,PETSC_MAX_PATH_LEN−1,file);
while (buf[0] == '％');

/* The first non-comment line has the matrix dimensions */
sscanf(buf,"%d %d %d\n",&m,&n,&nnz);
...
else
{
    for (i=0; i<nnz; i++) {
        fscanf(file, "%d %d %le %le\n", &row, &col, (double*)&re, (double*)&im);
        val = re + im * PETSC_i;
        row = row - 1; col = col - 1;
        ierr = MatSetValues(A, 1, &row, 1, &col, &val, INSERT_VALUES); CHKERRQ(ierr);
        if (isHermitian) {
            if (row != col) {
                ierr = MatSetValues(A, 1, &col, 1, &row, &val, INSERT_VALUES); CHKERRQ(ierr);
            }
        }
    }
}
fclose(file);
ierr = MatAssemblyBegin(A, MAT_FINAL ASSEMBLY); CHKERRQ(ierr);
ierr = MatAssemblyEnd(A, MAT_FINAL ASSEMBLY); CHKERRQ(ierr);
ierr = PetscPrintf(PETSC_COMM_SELF, "Reading matrix completes.\n"); CHKERRQ(ierr);
PetscViewer viewer;
ierr = PetscViewerBinaryOpen(PETSC_COMM_WORLD, fileout, FILE_MODE_WRITE, &viewer);
    CHKERRQ(ierr);
ierr = MatView(A, viewer); CHKERRQ(ierr);
ierr = PetscViewerDestroy(&viewer); CHKERRQ(ierr);
eigenvalue_parallel.c

/* Program usage: mpiexec ex1 [-help] [all PETSc options] */

static char help[] = "Solves a eigenvalue system in parallel.\n\n";

Main operation: Solve a eigenvalue problem in parallel
Input file format: Matrix market format (matrix and rhs in the same file)
Processor: 1 (sequential)

#include <petsctime.h>
#include "petscsys.h"
#include <slepceps.h>

#undef __FUNCT__
#define __FUNCT__ "main"

int main(int argc, char **args)
{
    Mat A;
    PetscInt i;
    PetscErrorCode ierr;
    char file [PETSC_MAX_PATH_LEN];
    PetscLogDouble numberOfFlops, tsolve1, tsolve2;
    EPS eps;     /* eigenproblem solver context */
const EPSType type;
PetscReal error, tol, re, im;
PetscScalar kr, ki;
Vec xr = 0, xi = 0;
PetscInt nev, maxit, its, nconv;
EPSWhich which;
EPSProblemType problemType;
PetscMPIInt rank;
PetscMPIInt numberOfProcessors;
PetscBool flg;
PetscBool isComplex;
PetscViewer fd;

SlepcInitialize(&argc, &args,(char*)0, help);

ient = MPI_Comm_rank(PETSC_COMM_WORLD, &rank); CHKERRQ(ierr);
ient = MPI_Comm_size(PETSC_COMM_WORLD, &numberOfProcessors); CHKERRQ(ierr);
ient = PetscOptionsGetString(PETSC_NULL, "-fin", file, PETSC_MAX_PATH_LEN, &flg);
    CHKERRQ(ierr);
    if (!flg) {
        SETERRQ(PETSC_COMM_WORLD, 1, "Must indicate matrix file with the -fin option");
    }
    /* Read file */
ient = PetscViewerBinaryOpen(PETSC_COMM_WORLD, file, FILE_MODE_READ, &fd); CHKERRQ(ierr);
    // Create matrix
ient = MatCreate(PETSC_COMM_WORLD, &A); CHKERRQ(ierr);
ient = MatSetFromOptions(A); CHKERRQ(ierr);
    // Load matrix from file
ient = MatLoad(A, fd); CHKERRQ(ierr);
    // Destroy viewer
// Assemble matrix
 ierr = MatAssemblyBegin(A, MAT_FINAL ASSEMBLY);CHKERRQ(ierr);
 ierr = MatAssemblyEnd(A, MAT_FINAL ASSEMBLY);CHKERRQ(ierr);

/* -----------------------------------------------------------
 Create the eigensolver and set various options
 ----------------------------------------------------------- */

/* Create eigensolver context */
 ierr = EPSCreate(PETSC_COMM_WORLD, &eps);CHKERRQ(ierr);

/* Set operators. In this case, it is a standard eigenvalue problem */
 ierr = EPSSetOperators(eps, A, PETSC_NULL);CHKERRQ(ierr);
 //ierr = EPSSetProblemType(eps, EPS_HEP);CHKERRQ(ierr);

/* Set solver parameters at runtime */
 ierr = EPSSetFromOptions(eps);CHKERRQ(ierr);

/* -----------------------------------------------------------
 Solve the eigensystem
 ----------------------------------------------------------- */
 PetscTime(tsolve1);
 ierr = EPSSolve(eps);CHKERRQ(ierr);
 PetscTime(tsolve2);

/* Optional: Get some information from the solver and display it */
 ierr = EPSGetProblemType(eps, &problemType);CHKERRQ(ierr);
ierr = EPSGetWhichEigenpairs(eps, &which); CHKERRQ(ierr);
ierr = EPSGetDimensions(eps,&nev,PETSC_NULL,PETSC_NULL); CHKERRQ(ierr);
ierr = EPSGetType(eps,&type); CHKERRQ(ierr);
ierr = EPSGetTolerances(eps,&tol,&maxit); CHKERRQ(ierr);
ierr = EPSGetConverged(eps,&nconv); CHKERRQ(ierr);
ierr = EPSGetIterationNumber(eps,&its); CHKERRQ(ierr);
ierr = PetscGetFlops(&numberOfFlops); CHKERRQ(ierr);

#if defined(PETSC_USE_COMPLEX)
isComplex = 1;
#else
isComplex = 0;
#endif

// Print output:
ierr = PetscPrintf(PETSC_COMM_WORLD,"%D\t%D\t%D\t%D\t%D\t%.4G\t%s\t%D\t%D\t%F\t%2.1e\t",isComplex,numberOfProcessors,problemType,which,nev,tol,type,nconv,its,numberOfFlops,(tsolve2-tsolve1)); CHKERRQ(ierr);

if (nconv>0) {
    for (i=0;i<nconv;i++) {
        /*
         Get converged eigenpairs: i-th eigenvalue is stored in kr (real part) and 
         ki (imaginary part)
        */
        ierr = EPSGetEigenpair(eps,i,&kr,&ki,xr,xi); CHKERRQ(ierr);
        /*
        Compute the relative error associated to each eigenpair
        */
        ierr = EPSComputeRelativeError(eps,i,&error); CHKERRQ(ierr);
    }
}

#if defined(PETSC_USE_COMPLEX)
    re = PetscRealPart(kr);
#endif
im = PetscImaginaryPart(kr);
#else
re = kr;
im = ki;
#endif

 ierr = PetscPrintf(PETSC_COMM_WORLD, ”%12G\t”, error); CHKERRQ(ierr);
}

 ierr = PetscPrintf(PETSC_COMM_WORLD, ”\n”); CHKERRQ(ierr);

// Destructors
 ierr = MatDestroy(&A); CHKERRQ(ierr);
  // ierr = PetscFinalize();
 ierr = SlepcFinalize(); CHKERRQ(ierr);
 return 0;
}
Appendix B

MATLAB Code - Obtains training data and generates Decision tree model

GetOptimizedSelection.m

```matlab
function GenerateCSV_OptimizedMultiSelectionBinRank()

% % Creates a csv file with training data required for machine learning
% % selects multiple best solvers
clc;

TestResults = importfile('Results_FinalCleanCopy.xlsx','Sheet1','D2:Q29326');

[m, n] = size(TestResults);

global featureCount;
featureCount = 8;
input = TestResults(1, 1:featureCount);
row = 1;

resultRowCounter = 1;
resultMatrix = {};

for i = 1:m
    if (isequal(TestResults(i, 1:featureCount), input) == 0)
        x = GetResult(A);
        if (size(x) > 0)
            [mx, nx] = size(x);
```

resultMatrix(resultRowCounter: resultRowCounter+mx-1,:) = x(:,:);
resultRowCounter=resultRowCounter+mx;
end
row=1;
A={};
input = TestResults(i,1:featureCount);
end
A(row,:) = TestResults(i,:);
row=row+1;
end
x = GetResult(A);
if(size(x)>0)
[mx,nx] = size(x);
resultMatrix(resultRowCounter: resultRowCounter+mx-1,:) = x(:,:);
resultRowCounter=resultRowCounter+mx;
end
featureCount=featureCount+1;
catgories = [false, true, false, true, false, true, false, true, true, true];
predNames = {'Size', 'Real', 'Processors', 'ProbType', 'Spectrum', 'NoOfEigvalues',
              'Tolerance', 'Binary', 'Rank'};
tc = ClassificationTree.fit(cell2mat(resultMatrix(:,1:featureCount)),resultMatrix(:,
                           featureCount+1), 'CategoricalPredictors', catgories, 'PredictorNames', predNames)
label = predict(tc, cell2mat(resultMatrix(:,1:featureCount))); view(tc, 'mode','graph')
view(tc, 'mode','text')
[E,SE,Nleaf,BestLevel] = cvLoss(tc)

% write training data to csv for orange
fid = fopen('ValidationSetSums.csv','wt');
columnNames = 'Id, Size, Real, Processors, ProbType, Spectrum, NoOfEigvalues, Tolerance, Binary, ExpectedSum, PredictedSum \\
';

RealColumnNumber =2;
ProbTypeColumnNumber =4;
SpectrumColumnNumber =5;
TolColumnNumber =7;

featureCount=featureCount−1; %excluding rank

input=resultMatrix(1,1:featureCount);
expectedSum=0;
predictedSum=0;
mispredict =0.0;
total =0.0;
countid =0;
fprintf(fid, columnNames);
for i=1:size(resultMatrix,1)
    if (isequal(resultMatrix(i,1:featureCount), input)==0)
        input{1,RealColumnNumber} = IsReal( input{1,RealColumnNumber});
        input{1,ProbTypeColumnNumber} = GetProbType( input{1,ProbTypeColumnNumber});
        input{1,SpectrumColumnNumber} = GetSpectrum( input{1,SpectrumColumnNumber});
        input{1,TolColumnNumber} = GetTolerancePower( input{1,TolColumnNumber});
        if (expectedSum˜=predictedSum)
            mispredict=mispredict+1;
        end
        total=total+1;
        countid=countid+1;
        fprintf(fid, '%d,%d,%s ,%d,%s ,%s ,%d,%d,%d,%d,%d \\
', countid , input{1,1:featureCount}, expectedSum, predictedSum);
        expectedSum=0;expectedSum, predictedSum);
        predictedSum=0;
    end
expectedSum = expectedSum + GetSolverNumber(resultMatrix(i, featureCount + 2));
predictedSum = predictedSum + GetSolverNumber(cell2mat(label(i, 1)));

input = resultMatrix(i, 1:featureCount);
end
fclose(fid);

display('Mispredict');
display(mispredict/total);

%% write training data to csv for orange
fid = fopen('TrainingSetForOrange.csv', 'wt');
columnNames = 'Size, Real, Processors, ProbType, Spectrum, NoOfEigvalues, Tolerance, Binary, Rank, Eigensolver \n';

RealColumnNumber = 2;
ProbTypeColumnNumber = 4;
SpectrumColumnNumber = 5;
TolColumnNumber = 7;
BinColumnNumber = 8;
RankColumnNumber = 9;

fprintf(fid, columnNames);
featureCount = featureCount + 1;
for i = 1:size(resultMatrix, 1)
    resultMatrix{i, RealColumnNumber} = IsReal(resultMatrix{i, RealColumnNumber});
    resultMatrix{i, ProbTypeColumnNumber} = GetProbType(resultMatrix{i, ProbTypeColumnNumber});
    resultMatrix{i, SpectrumColumnNumber} = GetSpectrum(resultMatrix{i, SpectrumColumnNumber});
    resultMatrix{i, TolColumnNumber} = GetTolerancePower(resultMatrix{i, TolColumnNumber});
resultMatrix{i,BinColumnNumber} = IsBinary(resultMatrix{i,BinColumnNumber});
resultMatrix{i,RankColumnNumber} = GetRank(resultMatrix{i,RankColumnNumber});

fprintf(fid, '%d,%s,%d,%s,%s,%d,%d,%s
', resultMatrix{i,1:featureCount+1});
end
fclose(fid);
end

function [rankString] = GetRank(rankValue)
  rankString = 'First';
  if(rankValue==2)
    rankString = 'Second';
  elseif(rankValue==3)
    rankString = 'Third';
  elseif(rankValue==4)
    rankString = 'Fourth';
  elseif(rankValue==5)
    rankString = 'Fifth';
  elseif(rankValue==6)
    rankString = 'Sixth';
  end
end

function [number] = GetSolverNumber(solver)
solver=strtrim(solver);
number=0;
if(strcmp(solver,'power'))
  number=.001;
elseif(strcmp(solver,'subspace'))
  number=0.01;
elseif(strcmp(solver,'arnoldi'))

number = 0.1;

elseif (strcmp(solver, 'lanczos'))
    number = 1;
elseif (strcmp(solver, 'krylovshur'))
    number = 10;
elseif (strcmp(solver, 'gd'))
    number = 100;
elseif (strcmp(solver, 'jd'))
    number = 1000;
end
end

function [binString] = IsBinary(binValue)
    binString = 'No';
    if (binValue == 1)
        binString = 'Yes';
    end
end

function [tolPower] = GetTolerancePower(tolValue)
    tolPower = log10(tolValue);
end

function [spectrumString] = GetSpectrum(spectrumValue)
    spectrumString = 'LargestMagnitude';
    if (spectrumValue == 2)
        spectrumString = 'SmallestMagnitude';
    elseif (spectrumValue == 3)
        spectrumString = 'LargestReal';
    elseif (spectrumValue == 4)
        spectrumString = 'SmallestReal';
    elseif (spectrumValue == 5)
        spectrumString = 'LargestImaginary';
    end
end
178     elseif (spectrumValue==6)
179         spectrumString = 'SmallestImaginary';
180     end
181 end
182
183 function [probTypeString] = GetProbType(probTypeValue)
184     probTypeString = 'Hermitian';
185     if (probTypeValue==3)
186         probTypeString = 'Non-Hermitian';
187     end
188 end
189
190 function [realString] = IsReal(realValue)
191     realString = 'Complex';
192     if (realValue==0)
193         realString = 'Real';
194     end
195 end
196
197 function [resultVec] = GetResult(A)
198 global featureCount;
199     inputVector = cell2mat(A(1,2:featureCount)); % get features = from SIZE till TOLERANCE
200     ASub= A(:,featureCount+1:end);
201     %input vector column numbers
202     eigenCol = 5;
203     tolCol = 6;
204
205     %ASub column numbers
206     convergedEigenPairsCol = 2;
207     hasConvCol = 6;
timeCol = 5;

% check residual with tolerance
vec = cell2mat(ASub(:, hasConvCol));
rm = find(~vec);
ASub = removerows(ASub, 'ind', rm);

[m, n] = size(ASub);

if (m > 1)
    % sort in terms of time taken and return the solver
    ASub = sortrows(ASub, timeCol);
    % remove rows that are bad, takes more than 10% time of min
    vec = cell2mat(ASub(:, timeCol));
    rm = find(vec > 1.1 * vec(1)); % if time taken is more than 10 percent min time
    ASub = removerows(ASub, 'ind', rm);

elseif (m == 0)
    % worst case if no results
    ASub = A(:, featureCount + 1:end);
    vec = cell2mat(ASub(:, convergedEigenPairsCol));
    rm = find(vec == 0); % change
    ASub = removerows(ASub, 'ind', rm);
    [m, n] = size(ASub);
    if (m > 0)
        ASub = sortrows(ASub, -convergedEigenPairsCol);
        vec = cell2mat(ASub(:, convergedEigenPairsCol));
        rm = find(vec < vec(1)); % if there exists other solvers with the max converged eigenvalues
        ASub = removerows(ASub, 'ind', rm);
    end

    if (~isempty(ASub))
        NConvVector = ASub(1,:);
    end

NConvVector{1,1} = 'NoConvergence';

ASub = [NConvVector ; ASub];

end

end

[m, n] = size(ASub);

if(m>0)
    combinedStr = strjoin(ASub(1:m,1)', ', ' - ');
    solvers = cellstr(combinedStr);
    if(~isempty(strfind(combinedStr, 'subspace')))
        display('is not empty);
        % solvers
        inputVector
        A(:, featureCount + 1:end)
        % m=1;
        ASub
        ASub(1,1) = cellstr('krylovshur');
        % ASub
        % A(:, featureCount + 1:end)
        % ASub(1:m,1)
    end

    rank = num2cell(1:m)';
    resultVec = horzcat(A(1:m,1:featureCount), rank, ASub(1:m,1));
else
    % display('Skipping no results')
    resultVec = []; 
    resultVec = horzcat(A(1,1:featureCount), {1}, 'NoConvergence');
end
end
Appendix C

MATLAB generated decision tree

```matlab
1  if Spectrum in {2 4} then node 2 elseif Spectrum in {1 3 5 6} then node 3 else krylovschur
2  if ProbType=1 then node 4 elseif ProbType=3 then node 5 else NoConvergence
3  if Spectrum in {1 3} then node 6 elseif Spectrum in {5 6} then node 7 else krylovschur
4  if Binary=0 then node 8 elseif Binary=1 then node 9 else NoConvergence
5  if Spectrum=2 then node 10 elseif Spectrum=4 then node 11 else krylovschur
6  if Binary=0 then node 12 elseif Binary=1 then node 13 else krylovschur
7  if Real=0 then node 14 elseif Real=1 then node 15 else arnoldi
8  if Rank=1 then node 16 elseif Rank=2 then node 17 else lanczos
9  if Rank=1 then node 18 elseif Rank in {2 3} then node 19 else NoConvergence
10 if Size<839 then node 20 elseif Size>=839 then node 21 else NoConvergence
11 if Rank=1 then node 22 elseif Rank=2 then node 23 else krylovschur
12 if ProbType=1 then node 24 elseif ProbType=3 then node 25 else krylovschur
13 if Rank=1 then node 26 elseif Rank in {2 3} then node 27 else NoConvergence
14 if Size<237736 then node 28 elseif Size>=237736 then node 29 else arnoldi
15 if Size<19847.5 then node 30 elseif Size>=19847.5 then node 31 else krylovschur
16 if NoOfEigvalues<3.5 then node 32 elseif NoOfEigvalues>=3.5 then node 33 else lanczos
17 if NoOfEigvalues<3.5 then node 34 elseif NoOfEigvalues>=3.5 then node 35 else lanczos
18 class = NoConvergence
```
if Tolerance < 5.0005e-05 then node 36 elif Tolerance >= 5.0005e-05 then node 37 else krylovshur
if Processors < 1.5 then node 38 elif Processors >= 1.5 then node 39 else krylovshur
if Size < 30987.5 then node 40 elif Size >= 30987.5 then node 41 else NoConvergence
if NoOfEigvalues < 1.5 then node 42 elif NoOfEigvalues >= 1.5 then node 43 else krylovshur
if Processors < 18 then node 44 elif Processors >= 18 then node 45 else arnoldi
if NoOfEigvalues < 3.5 then node 46 elif NoOfEigvalues >= 3.5 then node 47 else krylovshur
if Size < 30987.5 then node 48 elif Size >= 30987.5 then node 49 else krylovshur
if Size < 149402 then node 50 elif Size >= 149402 then node 51 else NoConvergence
if Rank = 2 then node 52 elif Rank = 3 then node 53 else krylovshur
if NoOfEigvalues < 1.5 then node 54 elif NoOfEigvalues >= 1.5 then node 55 else arnoldi
if NoOfEigvalues < 3.5 then node 56 elif NoOfEigvalues >= 3.5 then node 57 else arnoldi
if Tolerance < 5.0005e-05 then node 58 elif Tolerance >= 5.0005e-05 then node 59 else arnoldi
if NoOfEigvalues < 7.5 then node 60 elif NoOfEigvalues >= 7.5 then node 61 else krylovshur
if Size < 1599.5 then node 62 elif Size >= 1599.5 then node 63 else lanczos
if Tolerance < 5.0005e-05 then node 64 elif Tolerance >= 5.0005e-05 then node 65 else NoConvergence
if NoOfEigvalues < 1.5 then node 66 elif NoOfEigvalues >= 1.5 then node 67 else jd
if Size < 25786 then node 68 elif Size >= 25786 then node 69 else lanczos
if Spectrum = 2 then node 70 elif Spectrum = 4 then node 71 else krylovshur
if Spectrum = 2 then node 72 elif Spectrum = 4 then node 73 else krylovshur
if NoOfEigvalues < 3.5 then node 74 elif NoOfEigvalues >= 3.5 then node 75 else krylovshur
class = krylovshur
class = NoConvergence
if Size < 125888 then node 76 elseif Size >= 125888 then node 77 else NoConvergence
if Tolerance < 5.0005e-05 then node 78 elseif Tolerance >= 5.0005e-05 then node 79 else krylovshur
if Real = 0 then node 80 elseif Real = 1 then node 81 else krylovshur
if NoOfEigvalues < 3.5 then node 82 elseif NoOfEigvalues >= 3.5 then node 83 else krylovshur
if NoOfEigvalues < 1.5 then node 84 elseif NoOfEigvalues >= 1.5 then node 85 else arnoldi
if Processors < 6 then node 86 elseif Processors >= 6 then node 87 else krylovshur
if NoOfEigvalues < 7.5 then node 88 elseif NoOfEigvalues >= 7.5 then node 89 else krylovshur
if NoOfEigvalues < 1.5 then node 90 elseif NoOfEigvalues >= 1.5 then node 91 else krylovshur
if Size < 125888 then node 92 elseif Size >= 125888 then node 93 else krylovshur
if NoOfEigvalues < 1.5 then node 94 elseif NoOfEigvalues >= 1.5 then node 95 else NoConvergence
if Tolerance < 5.0005e-05 then node 96 elseif Tolerance >= 5.0005e-05 then node 97 else jd
if Tolerance < 5.05e-09 then node 98 elseif Tolerance >= 5.05e-09 then node 99 else krylovshur
class = krylovshur
if Tolerance < 5.0005e-05 then node 100 elseif Tolerance >= 5.0005e-05 then node 101 else arnoldi
if NoOfEigvalues < 7.5 then node 102 elseif NoOfEigvalues >= 7.5 then node 103 else arnoldi
if Spectrum = 5 then node 104 elseif Spectrum = 6 then node 105 else arnoldi
class = krylovshur
if Rank = 2 then node 106 elseif Rank in {1 3} then node 107 else arnoldi
if NoOfEigvalues < 3.5 then node 108 elseif NoOfEigvalues >= 3.5 then node 109 else krylovshur
if Spectrum = 5 then node 110 elseif Spectrum = 6 then node 111 else krylovshur
if Rank = 1 then node 112 elseif Rank = 2 then node 113 else krylovshur
class = lanczos
if Tolerance < 5.05e-09 then node 114 elseif Tolerance >= 5.05e-09 then node 115 else lanczos
if Size < 696 then node 116 elseif Size >= 696 then node 117 elseif NoConvergence
if Size < 25786 then node 118 elseif Size >= 25786 then node 119 elseif lanczos
if Processors < 3 then node 120 elseif Processors >= 3 then node 121 elseif gd
class = jd
if Size < 696 then node 122 elseif Size >= 696 then node 123 elseif lanczos
if Size < 93823 then node 124 elseif Size >= 93823 then node 125 elseif lanczos
if Size < 22204 then node 126 elseif Size >= 22204 then node 127 elseif krylovshur
if NoOfEigvalues < 7.5 then node 128 elseif NoOfEigvalues >= 7.5 then node 129 elseif krylovshur
if Size < 22204 then node 130 elseif Size >= 22204 then node 131 elseif krylovshur
if Rank = 2 then node 132 elseif Rank = 3 then node 133 elseif arnoldi
class = jd
class = krylovshur
class = jd
if Rank = 1 then node 134 elseif Rank = 2 then node 135 elseif NoConvergence
if Processors < 10 then node 136 elseif Processors >= 10 then node 137 elseif krylovshur
if Size < 839 then node 138 elseif Size >= 839 then node 139 elseif krylovshur
class = krylovshur
if NoOfEigvalues < 3.5 then node 140 elseif NoOfEigvalues >= 3.5 then node 141 elseif krylovshur
if Size < 839 then node 142 elseif Size >= 839 then node 143 elseif krylovshur
class = arnoldi
class = arnoldi
class = krylovshur
if Rank = 2 then node 144 elseif Rank in {1, 3} then node 145 elseif lanczos
if Size < 25786 then node 146 elseif Size >= 25786 then node 147 elseif krylovshur
if Size < 93823 then node 148 elseif Size >= 93823 then node 149 elseif krylovshur
if Real = 0 then node 150 elseif Real = 1 then node 151 elseif krylovshur
if Size < 839 then node 152 elseif Size >= 839 then node 153 elseif krylovshur
if Rank=1 then node 154 elseif Rank=2 then node 155 else krylovshur
if Spectrum=1 then node 156 elseif Spectrum=3 then node 157 else jd
if Rank=1 then node 158 elseif Rank=2 then node 159 else krylovshur
if Tolerance < 5.0005e−05 then node 160 elseif Tolerance >= 5.0005e−05 then node 161 else NoConvergence
class = NoConvergence
class = jd
if NoOfEigvalues < 1.5 then node 162 elseif NoOfEigvalues >= 1.5 then node 163 else jd
if NoOfEigvalues < 3.5 then node 164 elseif NoOfEigvalues >= 3.5 then node 165 else krylovshur
if NoOfEigvalues < 3.5 then node 166 elseif NoOfEigvalues >= 3.5 then node 167 else arnoldi
class = arnoldi
if Size < 11979 then node 168 elseif Size >= 11979 then node 169 else arnoldi
class = arnoldi
if Size < 11979 then node 170 elseif Size >= 11979 then node 171 else arnoldi
if NoOfEigvalues < 1.5 then node 172 elseif NoOfEigvalues >= 1.5 then node 173 else arnoldi
class = NoConvergence
if NoOfEigvalues < 1.5 then node 174 elseif NoOfEigvalues >= 1.5 then node 175 else arnoldi
if NoOfEigvalues < 3.5 then node 176 elseif NoOfEigvalues >= 3.5 then node 177 else NoConvergence
if Spectrum = 5 then node 178 elseif Spectrum = 6 then node 179 else krylovshur
if Spectrum = 5 then node 180 elseif Spectrum = 6 then node 181 else krylovshur
if NoOfEigvalues < 1.5 then node 182 elseif NoOfEigvalues >= 1.5 then node 183 else krylovshur
if NoOfEigvalues < 1.5 then node 184 elseif NoOfEigvalues >= 1.5 then node 185 else krylovshur
class = krylovshur
class = jd
if Size < 25786 then node 186 elseif Size >= 25786 then node 187 else NoConvergence
if Size < 25786 then node 188 elseif Size >= 25786 then node 189 else lanczos
if Processors < 72 then node 190 elseif Processors >= 72 then node 191 else jd
if Size < 25786 then node 192 elseif Size >= 25786 then node 193 else NoConvergence
if Size < 1599.5 then node 194 elseif Size >= 1599.5 then node 195 else lanczos
if Size < 93823 then node 196 elseif Size >= 93823 then node 197 else NoConvergence
class = gd
if Tolerance < 5.0005e-05 then node 198 elseif Tolerance >= 5.0005e-05 then node 199 else jd
if NoOfEigvalues < 7.5 then node 200 elseif NoOfEigvalues >= 7.5 then node 201 else lanczos
class = lanczos
if NoOfEigvalues < 7.5 then node 202 elseif NoOfEigvalues >= 7.5 then node 203 else jd
class = lanczos
if NoOfEigvalues < 3.5 then node 204 elseif NoOfEigvalues >= 3.5 then node 205 else krylovschur
class = krylovschur
if NoOfEigvalues < 3.5 then node 206 elseif NoOfEigvalues >= 3.5 then node 207 else krylovschur
class = krylovschur
if NoOfEigvalues < 3.5 then node 208 elseif NoOfEigvalues >= 3.5 then node 209 else krylovschur
class = krylovschur
if Size < 22204 then node 210 elseif Size >= 22204 then node 211 else arnoldi
class = krylovschur
class = NoConvergence
class = krylovschur
if Size < 12420 then node 212 elseif Size >= 12420 then node 213 else krylovschur
class = krylovschur
class = krylovschur
class = krylovschur
if Processors < 18 then node 214 elseif Processors >= 18 then node 215 else arnoldi
if Tolerance < 5.0005e-05 then node 216 elseif Tolerance >= 5.0005e-05 then node 217 else krylovschur
if Processors < 36 then node 218 elseif Processors >= 36 then node 219 else krylovshur
class = arnoldi
if Size < 237736 then node 220 elseif Size >= 237736 then node 221 else krylovshur
if Real = 0 then node 222 elseif Real = 1 then node 223 else krylovshur
if Size < 3359.5 then node 224 elseif Size >= 3359.5 then node 225 else lanczos
if Rank = 3 then node 226 elseif Rank in {1 2 4} then node 227 else krylovshur
if Rank = 1 then node 228 elseif Rank in {2 3} then node 229 else krylovshur
if Size < 3359.5 then node 230 elseif Size >= 3359.5 then node 231 else krylovshur
class = krylovshur
if Rank = 1 then node 232 elseif Rank = 2 then node 233 else krylovshur
if Tolerance < 5.05e-09 then node 234 elseif Tolerance >= 5.05e-09 then node 235 else krylovshur
class = krylovshur
if Processors < 18 then node 236 elseif Processors >= 18 then node 237 else krylovshur
if Real = 0 then node 238 elseif Real = 1 then node 239 else krylovshur
if Processors < 3 then node 240 elseif Processors >= 3 then node 241 else krylovshur
if NoOfEigvalues < 7.5 then node 242 elseif NoOfEigvalues >= 7.5 then node 243 else krylovshur
class = jd
class = krylovshur
class = arnoldi
class = NoConvergence
if Spectrum = 1 then node 244 elseif Spectrum = 3 then node 245 else NoConvergence
class = krylovshur
if NoOfEigvalues < 3.5 then node 246 elseif NoOfEigvalues >= 3.5 then node 247 else jd
class = krylovshur
if Spectrum = 1 then node 248 elseif Spectrum = 3 then node 249 else krylovshur
if Size < 149402 then node 250 elseif Size >= 149402 then node 251 else arnoldi
if Size < 22204 then node 252 elseif Size >= 22204 then node 253 else krylovshur
if Processors < 144 then node 254 elseif Processors ≥ 144 then node 255 else krylovshur

class = arnoldi

if Spectrum = 5 then node 256 elseif Spectrum = 6 then node 257 else arnoldi

if Size < 118460 then node 258 elseif Size ≥ 118460 then node 259 else arnoldi
class = arnoldi

if Processors < 30 then node 260 elseif Processors ≥ 30 then node 261 else arnoldi
class = jd
class = arnoldi

if Spectrum = 5 then node 262 elseif Spectrum = 6 then node 263 else arnoldi

if Rank = 1 then node 264 elseif Rank = 3 then node 265 else NoConvergence

if NoOfEigvalues < 1.5 then node 266 elseif NoOfEigvalues ≥ 1.5 then node 267 else krylovshur
class = krylovshur

if NoOfEigvalues < 7.5 then node 268 elseif NoOfEigvalues ≥ 7.5 then node 269 else jd
class = krylovshur

if Tolerance < 5.05e−09 then node 270 elseif Tolerance ≥ 5.05e−09 then node 271 else krylovshur

if Tolerance < 5.0005e−05 then node 272 elseif Tolerance ≥ 5.0005e−05 then node 273 else jd

if Tolerance < 5.0005e−05 then node 274 elseif Tolerance ≥ 5.0005e−05 then node 275 else arnoldi

if Tolerance < 5.0005e−05 then node 276 elseif Tolerance ≥ 5.0005e−05 then node 277 else krylovshur

if NoOfEigvalues < 1.5 then node 278 elseif NoOfEigvalues ≥ 1.5 then node 279 else NoConvergence

if Size < 93823 then node 280 elseif Size ≥ 93823 then node 281 else NoConvergence

if Size < 3359.5 then node 282 elseif Size ≥ 3359.5 then node 283 else lanczos

if Size < 93823 then node 284 elseif Size ≥ 93823 then node 285 else lanczos

if Tolerance < 5.05e−09 then node 286 elseif Tolerance ≥ 5.05e−09 then node 287 else jd
if NoOfEigvalues < 7.5 then node 288 elseif NoOfEigvalues >= 7.5 then node 289 else
    NoConvergence
class = NoConvergence
if Size < 93823 then node 290 elseif Size >= 93823 then node 291 else NoConvergence
if NoOfEigvalues < 7.5 then node 292 elseif NoOfEigvalues >= 7.5 then node 293 else lanczos
if Size < 3359.5 then node 294 elseif Size >= 3359.5 then node 295 else lanczos
class = gd
class = NoConvergence
if Processors < 18 then node 296 elseif Processors >= 18 then node 297 else gd
class = jd
class = lanczos
class = jd
class = jd
if Processors < 6 then node 298 elseif Processors >= 6 then node 299 else gd
class = krylovshur
if Processors < 10 then node 300 elseif Processors >= 10 then node 301 else krylovshur
if Size < 22204 then node 302 elseif Size >= 22204 then node 303 else krylovshur
if Tolerance < 5.05e-09 then node 304 elseif Tolerance >= 5.05e-09 then node 305 else arnoldi
if NoOfEigvalues < 1.5 then node 306 elseif NoOfEigvalues >= 1.5 then node 307 else krylovshur
class = krylovshur
if NoOfEigvalues < 7.5 then node 308 elseif NoOfEigvalues >= 7.5 then node 309 else krylovshur
class = arnoldi
class = krylovshur
if Size < 30987.5 then node 310 elseif Size >= 30987.5 then node 311 else krylovshur
if Size < 125888 then node 312 elseif Size >= 125888 then node 313 else arnoldi
class = krylovshur
class = krylovshur
if Size < 10847.5 then node 314 else Size >= 19847.5 then node 315 else arnoldi

class = krylovshur

if NoOfEigvalues < 7.5 then node 316 else NoOfEigvalues >= 7.5 then node 317 else krylovshur

if NoOfEigvalues < 1.5 then node 318 else NoOfEigvalues >= 1.5 then node 319 else krylovshur

class = arnoldi

if Size < 93823 then node 320 else Size >= 93823 then node 321 else krylovshur

if Tolerance < 5.05e-09 then node 322 else Tolerance >= 5.05e-09 then node 323 else arnoldi

if Processors < 3 then node 324 else Processors >= 3 then node 325 else lanczos

if NoOfEigvalues < 1.5 then node 326 else NoOfEigvalues >= 1.5 then node 327 else lanczos

if Tolerance < 5.05e-09 then node 328 else Tolerance >= 5.05e-09 then node 329 else lanczos

if Size < 1599.5 then node 330 else Size >= 1599.5 then node 331 else krylovshur

if NoOfEigvalues < 1.5 then node 332 else NoOfEigvalues >= 1.5 then node 333 else lanczos

if NoOfEigvalues < 1.5 then node 334 else NoOfEigvalues >= 1.5 then node 335 else krylovshur

if Tolerance < 5.0005e-05 then node 336 else Tolerance >= 5.0005e-05 then node 337 else krylovshur

if Processors < 18 then node 338 else Processors >= 18 then node 339 else lanczos

if Processors < 6 then node 340 else Processors >= 6 then node 341 else krylovshur

if Tolerance < 5.0005e-05 then node 342 else Tolerance >= 5.0005e-05 then node 343 else lanczos

class = arnoldi

if Tolerance < 5.0005e-05 then node 344 else Tolerance >= 5.0005e-05 then node 345 else krylovshur

if Size < 12420 then node 346 else Size >= 12420 then node 347 else arnoldi

if Size < 12420 then node 348 else Size >= 12420 then node 349 else krylovshur

class = krylovshur
if NoOfEigvalues < 3.5 then node 350 elif NoOfEigvalues >= 3.5 then node 351 else krylovschur

class = krylovschur

if Tolerance < 5.05e-09 then node 352 elif Tolerance >= 5.05e-09 then node 353

else arnoldi

if NoOfEigvalues < 1.5 then node 354 elif NoOfEigvalues >= 1.5 then node 355 else krylovschur

class = krylovschur

if Size < 22204 then node 356 elif Size >= 22204 then node 357 else arnoldi

class = NoConvergence

class = jd

class = arnoldi

if Size < 22204 then node 358 elif Size >= 22204 then node 359 else arnoldi

if NoOfEigvalues < 7.5 then node 360 elif NoOfEigvalues >= 7.5 then node 361 else krylovschur

if Tolerance < 5.0005e-05 then node 362 elif Tolerance >= 5.0005e-05 then node 363 else arnoldi

class = gd

if Spectrum = 1 then node 364 elif Spectrum = 3 then node 365 else krylovschur

if Size < 149402 then node 366 elif Size >= 149402 then node 367 else arnoldi

class = krylovschur

class = arnoldi

class = arnoldi

if Tolerance < 5.0005e-05 then node 368 elif Tolerance >= 5.0005e-05 then node 369 else arnoldi

class = arnoldi

if Processors < 6 then node 370 elif Processors >= 6 then node 371 else arnoldi

class = arnoldi

if Rank = 1 then node 372 elif Rank = 2 then node 373 else arnoldi

if Processors < 36 then node 374 elif Processors >= 36 then node 375 else arnoldi

if Tolerance < 5.05e-09 then node 376 elif Tolerance >= 5.05e-09 then node 377 else arnoldi
if NoOfEigvalues < 7.5 then node 378
elseif NoOfEigvalues >= 7.5 then node 379
else
    NoConvergence
end

if Processors < 36 then node 380
elseif Processors >= 36 then node 381
else
    krylovSchur
end

class = krylovSchur

class = jd

class = arnoldi

class = arnoldi

if Rank = 1 then node 382
elseif Rank = 2 then node 383
else
    krylovSchur
end

if NoOfEigvalues < 3.5 then node 384
elseif NoOfEigvalues >= 3.5 then node 385
else
    jd
end

class = jd

if Tolerance < 5.05e-09 then node 386
elseif Tolerance >= 5.05e-09 then node 387
else
    arnoldi
end

class = krylovSchur

class = krylovSchur

if NoOfEigvalues < 3.5 then node 388
elseif NoOfEigvalues >= 3.5 then node 389
else
    arnoldi
end

class = NoConvergence

if Size < 3359.5 then node 390
elseif Size >= 3359.5 then node 391
else
    NoConvergence
end

if Processors < 36 then node 392
elseif Processors >= 36 then node 393
else
    jd
end

class = NoConvergence

if Tolerance < 5.0005e-05 then node 394
elseif Tolerance >= 5.0005e-05 then node 395
else
    lanczos
end

class = lanczos

if Processors < 18 then node 396
elseif Processors >= 18 then node 397
else
    gd
end

class = lanczos

if Processors < 10 then node 398
elseif Processors >= 10 then node 399
else
    jd
end

class = jd

class = NoConvergence

class = krylovSchur
if Processors < 36 then node 400 elseif Processors >= 36 then node 401 else jd

class = NoConvergence

class = lanczos

if Size < 696 then node 402 elseif Size >= 696 then node 403 else lanczos

class = NoConvergence

class = lanczos

class = gd

class = jd

class = jd

class = gd

class = krylovschur

class = arnoldi

if Processors < 6 then node 404 elseif Processors >= 6 then node 405 else

krylovschur

class = krylovschur

class = krylovschur

class = arnoldi

class = krylovschur

class = arnoldi

if Processors < 10 then node 406 elseif Processors >= 10 then node 407 else

krylovschur

class = arnoldi

class = arnoldi

class = krylovschur

class = arnoldi

class = krylovschur

class = arnoldi

class = krylovschur

class = arnoldi

class = krylovschur

class = arnoldi
if Processors < 3 then node 408 elseif Processors >= 3 then node 409 else krylovshur

class = lanczos

class = arnoldi

class = krylovshur

if Size < 696 then node 410 elseif Size >= 696 then node 411 else lanczos

if Rank = 1 then node 412 elseif Rank = 3 then node 413 else lanczos

if Size < 25786 then node 414 elseif Size >= 25786 then node 415 else lanczos

if Size < 93823 then node 416 elseif Size >= 93823 then node 417 else lanczos

if Processors < 72 then node 418 elseif Processors >= 72 then node 419 else krylovshur

class = lanczos

if Rank = 1 then node 420 elseif Rank = 2 then node 421 else krylovshur

if NoOfEigvalues < 1.5 then node 422 elseif NoOfEigvalues >= 1.5 then node 423 else krylovshur

if Processors < 72 then node 424 elseif Processors >= 72 then node 425 else lanczos

if Processors < 10 then node 426 elseif Processors >= 10 then node 427 else krylovshur

if Spectrum = 1 then node 428 elseif Spectrum = 3 then node 429 else arnoldi

if Size < 93823 then node 430 elseif Size >= 93823 then node 431 else krylovshur

if Rank = 1 then node 432 elseif Rank = 2 then node 433 else krylovshur

if Size < 1599.5 then node 434 elseif Size >= 1599.5 then node 435 else krylovshur

if Rank in {1, 2} then node 436 elseif Rank = 3 then node 437 else lanczos

if Rank in {1, 2} then node 438 elseif Rank = 3 then node 439 else krylovshur

if Tolerance < 5.0005e-05 then node 440 elseif Tolerance >= 5.0005e-05 then node 441 else krylovshur

class = krylovshur

class = lanczos

class = krylovshur

class = krylovshur

if Processors < 3 then node 442 elseif Processors >= 3 then node 443 else krylovshur
if Tolerance < 5.0005e-05 then node 444 else Tolerance >= 5.0005e-05 then node 445
else krylovshur

if Rank = 1 then node 446 elseif Rank = 2 then node 447 else arnoldi

if Processors < 72 then node 448 elseif Processors >= 72 then node 449 else
krylovshur

class = krylovshur

if Tolerance < 5.0005e-05 then node 450 elseif Tolerance >= 5.0005e-05 then node 451
else krylovshur

class = krylovshur

class = krylovshur

if Spectrum = 1 then node 452 elseif Spectrum = 3 then node 453 else arnoldi

if Tolerance < 5.0005e-05 then node 454 elseif Tolerance >= 5.0005e-05 then node 455
else krylovshur

if NoOfEigvalues < 3.5 then node 456 elseif NoOfEigvalues >= 3.5 then node 457 else
jd

class = power

class = arnoldi

class = arnoldi

if Processors < 1.5 then node 458 elseif Processors >= 1.5 then node 459 else
krylovshur

if Processors < 72 then node 460 elseif Processors >= 72 then node 461 else arnoldi

if Processors < 144 then node 462 elseif Processors >= 144 then node 463 else
krylovshur

if Processors < 144 then node 464 elseif Processors >= 144 then node 465 else
arnoldi

if Spectrum = 1 then node 466 elseif Spectrum = 3 then node 467 else arnoldi

class = krylovshur

if Tolerance < 5.0005e-05 then node 468 elseif Tolerance >= 5.0005e-05 then node 469
else krylovshur

if Spectrum = 1 then node 470 elseif Spectrum = 3 then node 471 else arnoldi

if Spectrum = 1 then node 472 elseif Spectrum = 3 then node 473 else jd

if Processors < 3 then node 474 elseif Processors >= 3 then node 475 else arnoldi

class = arnoldi
class = arnoldi
if Processors < 10 then node 476 elseif Processors >= 10 then node 477 else arnoldi
class = krylovshur
class = arnoldi
if Processors < 18 then node 478 elseif Processors >= 18 then node 479 else jd
class = arnoldi
if Processors < 36 then node 480 elseif Processors >= 36 then node 481 else arnoldi
class = arnoldi
if Spectrum = 5 then node 482 elseif Spectrum = 6 then node 483 else NoConvergence
class = NoConvergence
if Rank = 1 then node 484 elseif Rank = 2 then node 485 else arnoldi
class = krylovshur
class = krylovshur
class = arnoldi
if Rank = 1 then node 486 elseif Rank = 2 then node 487 else jd
class = krylovshur
if Rank = 1 then node 488 elseif Rank = 2 then node 489 else arnoldi
class = arnoldi
class = krylovshur
class = arnoldi
class = NoConvergence
if Processors < 18 then node 490 elseif Processors >= 18 then node 491 else lanczos
class = jd
class = gd
class = NoConvergence
class = lanczos
if Processors < 1.5 then node 492 elseif Processors >= 1.5 then node 493 else gd
class = gd
if NoOfEigvalues < 7.5 then node 494 elseif NoOfEigvalues >= 7.5 then node 495 else krylovshur
class = jd
class = jd
if NoOfEigvalues < 7.5 then node 496 elseif NoOfEigvalues >= 7.5 then node 497 else gd
if Processors < 72 then node 498 elseif Processors >= 72 then node 499 else jd
class = lanczos
class = krylovschur
if Rank = 2 then node 500 elseif Rank = 3 then node 501 elseif arnoldi
class = krylovschur
class = arnoldi
if Tolerance < 5.0005e-05 then node 502 elseif Tolerance >= 5.0005e-05 then node 503 elseif krylovschur
if Tolerance < 5.05e-09 then node 504 elseif Tolerance >= 5.05e-09 then node 505 elseif krylovschur
if Spectrum = 1 then node 506 elseif Spectrum = 3 then node 507 elseif lanczos
if Tolerance < 5.0005e-05 then node 508 elseif Tolerance >= 5.0005e-05 then node 509 elseif lanczos
if Size < 696 then node 510 elseif Size >= 696 then node 511 elseif krylovschur
class = lanczos
class = lanczos
if Spectrum = 1 then node 512 elseif Spectrum = 3 then node 513 elseif lanczos
if Rank = 1 then node 514 elseif Rank = 3 then node 515 elseif lanczos
class = lanczos
if Tolerance < 5.05e-09 then node 516 elseif Tolerance >= 5.05e-09 then node 517 elseif lanczos
class = krylovschur
class = arnoldi
if NoOfEigvalues < 1.5 then node 518 elseif NoOfEigvalues >= 1.5 then node 519 elseif arnoldi
if Tolerance < 5.0005e-05 then node 520 elseif Tolerance >= 5.0005e-05 then node 521 elseif krylovschur
if Tolerance < 5.0005e-05 then node 522 elseif Tolerance >= 5.0005e-05 then node 523 elseif krylovschur
if Size < 3359.5 then node 524 elseif Size >= 3359.5 then node 525 elseif krylovschur
class = lanczos
if Tolerance < 5.0005e-05 then node 526 else Tolerance >= 5.0005e-05 then node 527

else krylovshur

class = lanczos

if Size < 93823 then node 528 elseif Size >= 93823 then node 529 else krylovshur

if Processors < 18 then node 530 elseif Processors >= 18 then node 531 else arnoldi

if Processors < 144 then node 532 elseif Processors >= 144 then node 533 else krylovshur

class = krylovshur

if Processors < 18 then node 534 elseif Processors >= 18 then node 535 else krylovshur

if Real = 0 then node 536 elseif Real = 1 then node 537 else krylovshur

if Size < 1599.5 then node 538 elseif Size >= 1599.5 then node 539 else arnoldi

if Size < 696 then node 540 elseif Size >= 696 then node 541 else arnoldi

if Processors < 3 then node 542 elseif Processors >= 3 then node 543 else krylovshur

if Processors < 6 then node 544 elseif Processors >= 6 then node 545 else lanczos

class = krylovshur

if Size < 25786 then node 546 elseif Size >= 25786 then node 547 else krylovshur

class = arnoldi

class = krylovshur

if Size < 3359.5 then node 548 elseif Size >= 3359.5 then node 549 else krylovshur

class = lanczos

if Rank = 1 then node 550 elseif Rank = 2 then node 551 else krylovshur

if Rank = 1 then node 552 elseif Rank = 2 then node 553 else krylovshur

if Rank = 1 then node 554 elseif Rank = 2 then node 555 else arnoldi

if Spectrum = 1 then node 556 elseif Spectrum = 3 then node 557 else arnoldi

class = krylovshur

if Rank = 1 then node 558 elseif Rank = 2 then node 559 else krylovshur

class = arnoldi

class = krylovshur

if Processors < 10 then node 560 elseif Processors >= 10 then node 561 else arnoldi

class = arnoldi

if Processors < 6 then node 562 elseif Processors >= 6 then node 563 else arnoldi
if Rank=1 then node 564 else Rank=2 then node 565 else krylovshur

class = krylovshur

if Rank=1 then node 566 else Rank=2 then node 567 else jd

if Tolerance < 5.0005e-05 then node 568 else Tolerance >= 5.0005e-05 then node

else krylovshur

class = krylovshur

if Processors < 3 then node 570 else Processors >= 3 then node 571 else arnoldi

if Processors < 6 then node 572 else Processors >= 6 then node 573 else arnoldi

class = krylovshur

class = krylovshur

class = arnoldi

if NoOfEigvalues < 1.5 then node 574 else NoOfEigvalues >= 1.5 then node 575 else

arnoldi

class = krylovshur

if Size < 22204 then node 576 else Size >= 22204 then node 577 else arnoldi

class = arnoldi

class = krylovshur

if NoOfEigvalues < 7.5 then node 578 else NoOfEigvalues >= 7.5 then node 579 else

arnoldi

if Tolerance < 5.0005e-05 then node 580 else Tolerance >= 5.0005e-05 then node

else arnoldi

if Tolerance < 5.0005e-05 then node 582 else Tolerance >= 5.0005e-05 then node

else krylovshur

class = jd

class = krylovshur

class = arnoldi

if Processors < 18 then node 584 else Processors >= 18 then node 585 else gd

if Rank=1 then node 586 else Rank=2 then node 587 else arnoldi

class = arnoldi

if Processors < 6 then node 588 else Processors >= 6 then node 589 else jd

class = NoConvergence

class = arnoldi

class = NoConvergence
if Processors < 36 then node 590  

elseif Processors ≥ 36 then node 591  

    NoConvergence  

class = NoConvergence  

class = arnoldi  

class = krylovshur  

class = jd  

class = krylovshur  

class = arnoldi  

class = krylovshur  

class = NoConvergence  

class = lanczos  

class = jd  

if Processors < 10 then node 592  

elseif Processors ≥ 10 then node 593  

else gd  

class = jd  

class = krylovshur  

if Tolerance < 5.05e−09 then node 594  

elseif Tolerance ≥ 5.05e−09 then node 595  

else gd  

class = gd  

class = jd  

class = krylovshur  

class = arnoldi  

class = krylovshur  

if NoOfEigvalues < 1.5 then node 596  

elseif NoOfEigvalues ≥ 1.5 then node 597  

else krylovshur  

class = krylovshur  

class = krylovshur  

if Size < 1015.5 then node 598  

elseif Size ≥ 1015.5 then node 599  

else lanczos  

if Processors < 1.5 then node 600  

elseif Processors ≥ 1.5 then node 601  

else lanczos  

if Rank = 1 then node 602  

elseif Rank = 3 then node 603  

else lanczos  

if Real = 0 then node 604  

elseif Real = 1 then node 605  

else lanczos  

class = lanczos
if Tolerance $< 5.0005 \times 10^{-05}$ then node 606  

else Tolerance $\geq 5.0005 \times 10^{-05}$ then node 607  

else arnoldi

if Real $= 0$ then node 608  

elseif Real $= 1$ then node 609  

else krylovshur

class = power

class = lanczos

class = lanczos

class = arnoldi

class = krylovshur

class = lanczos

if Processors $< 10$ then node 610  

elseif Processors $\geq 10$ then node 611  

else arnoldi

if Size $< 696$ then node 612  

elseif Size $\geq 696$ then node 613  

else lanczos

if Spectrum $= 1$ then node 614  

elseif Spectrum $= 3$ then node 615  

else krylovshur

if Processors $< 18$ then node 616  

elseif Processors $\geq 18$ then node 617  

else krylovshur

if Tolerance $< 5.05 \times 10^{-09}$ then node 618  

elseif Tolerance $\geq 5.05 \times 10^{-09}$ then node 619  

else krylovshur

if Processors $< 10$ then node 620  

elseif Processors $\geq 10$ then node 621  

else krylovshur

class = krylovshur

class = lanczos

class = krylovshur

class = krylovshur

if Tolerance $< 5.05 \times 10^{-09}$ then node 622  

elseif Tolerance $\geq 5.05 \times 10^{-09}$ then node 623  

else krylovshur

if Processors $< 72$ then node 626  

elseif Processors $\geq 72$ then node 627  

else arnoldi

if Rank $= 2$ then node 628  

elseif Rank $= 3$ then node 629  

else krylovshur

class = krylovshur

class = arnoldi

if Rank $= 2$ then node 630  

elseif Rank $= 3$ then node 631  

else krylovshur

class = krylovshur
if Processors < 72 then node 632 elseif Processors >= 72 then node 633 else krylovshur

class = arnoldi

class = krylovshur

if Rank = 1 then node 634 elseif Rank = 2 then node 635 else arnoldi

if Processors < 6 then node 636 elseif Processors >= 6 then node 637 else krylovshur

class = lanczos

class = krylovshur

class = lanczos

if Size < 25786 then node 638 elseif Size >= 25786 then node 639 else lanczos

if Tolerance < 5.0005e-05 then node 640 elseif Tolerance >= 5.0005e-05 then node 641 else krylovshur

if Rank = 1 then node 642 elseif Rank = 2 then node 643 else krylovshur

class = krylovshur

if Size < 93823 then node 644 elseif Size >= 93823 then node 645 else lanczos

class = krylovshur

class = lanczos

if Spectrum = 1 then node 646 elseif Spectrum = 3 then node 647 else krylovshur

class = arnoldi

class = arnoldi

class = krylovshur

class = arnoldi

if Processors < 3 then node 648 elseif Processors >= 3 then node 649 else arnoldi

if Tolerance < 5.0005e-05 then node 650 elseif Tolerance >= 5.0005e-05 then node 651 else krylovshur

class = arnoldi

class = arnoldi

class = krylovshur

class = krylovshur

class = arnoldi

if Processors < 3 then node 652 elseif Processors >= 3 then node 653 else arnoldi

class = krylovshur
if Processors < 6 then node 654 elseif Processors >= 6 then node 655 else krylovshur

class = krylovshur

class = arnoldi

class = krylovshur

class = arnoldi

if Size < 22204 then node 656 elseif Size >= 22204 then node 657 else arnoldi

if Spectrum=1 then node 658 elseif Spectrum=3 then node 659 else arnoldi

class = krylovshur

class = arnoldi

class = krylovshur

class = arnoldi

if NoOfEigvalues < 7.5 then node 660 elseif NoOfEigvalues >= 7.5 then node 661 else arnoldi

class = krylovshur

if NoOfEigvalues < 7.5 then node 662 elseif NoOfEigvalues >= 7.5 then node 663 else arnoldi

class = arnoldi

class = gd

class = NoConvergence

class = arnoldi

class = jd

if Tolerance < 5.0005e-05 then node 664 elseif Tolerance >= 5.0005e-05 then node 665 else gd

class = gd

class = jd
99

595 class = gd
596 class = krylovshur
597 class = lanczos
598 class = arnoldi
599 if Tolerance < 5.0005e-05 then node 666 elseif Tolerance > = 5.0005e-05 then node 667 else lanczos
600 class = power
601 class = lanczos
602 class = lanczos
603 class = arnoldi
604 class = lanczos
605 if Tolerance < 5.05e-09 then node 668 elseif Tolerance > = 5.05e-09 then node 669 else krylovshur
606 class = arnoldi
607 class = krylovshur
608 if NoOfEigvalues < 1.5 then node 670 elseif NoOfEigvalues > = 1.5 then node 671 else krylovshur
609 if Tolerance < 5.05e-09 then node 672 elseif Tolerance > = 5.05e-09 then node 673 else lanczos
610 class = krylovshur
611 if Processors < 36 then node 674 elseif Processors > = 36 then node 675 else arnoldi
612 if Tolerance < 5.0005e-05 then node 676 elseif Tolerance > = 5.0005e-05 then node 677 else lanczos
613 if Processors < 10 then node 678 elseif Processors > = 10 then node 679 else arnoldi
614 if Tolerance < 5.05e-09 then node 680 elseif Tolerance > = 5.05e-09 then node 681 else krylovshur
615 if Processors < 18 then node 682 elseif Processors > = 18 then node 683 else krylovshur
616 if Size < 696 then node 684 elseif Size > = 696 then node 685 else krylovshur
617 class = krylovshur
618 if Size < 3359.5 then node 686 elseif Size > = 3359.5 then node 687 else krylovshur
619 if Size < 3359.5 then node 688 elseif Size > = 3359.5 then node 689 else lanczos
620 class = krylovshur
if Size < 3359.5 then node 690 elseif Size >= 3359.5 then node 691 else arnoldi
krylovschur
if Processors < 10 then node 692 elseif Processors >= 10 then node 693 else
arnoldi
if Rank = 1 then node 694 elseif Rank in {2, 4} then node 695 else krylovschur
class = krylovschur
if Processors < 18 then node 696 elseif Processors >= 18 then node 697 else lanczos
if Processors < 36 then node 698 elseif Processors >= 36 then node 699 else arnoldi
if Tolerance < 5.05e-09 then node 700 elseif Tolerance >= 5.05e-09 then node 701
else arnoldi
if Tolerance < 5.05e-09 then node 702 elseif Tolerance >= 5.05e-09 then node 703
else krylovschur
if Tolerance < 5.05e-09 then node 704 elseif Tolerance >= 5.05e-09 then node 705
else krylovschur
if Processors < 36 then node 706 elseif Processors >= 36 then node 707 else arnoldi
if Processors < 144 then node 708 elseif Processors >= 144 then node 709 else
krylovschur
class = krylovschur
class = arnoldi
if Processors < 1.5 then node 710 elseif Processors >= 1.5 then node 711 else
arnoldi
if Processors < 16 then node 712 elseif Processors >= 16 then node 713 else
krylovschur
class = lanczos
if Rank = 1 then node 714 elseif Rank in {2, 3} then node 715 else arnoldi
if Tolerance < 5.0005e-05 then node 716 elseif Tolerance >= 5.0005e-05 then node
717 else lanczos
class = lanczos
if Rank = 1 then node 718 elseif Rank = 2 then node 719 else krylovschur
if Rank = 1 then node 720 elseif Rank = 2 then node 721 else krylovschur
class = krylovschur
class = lanczos
class = lanczos
class = krylovschur
class = krylovshur

if Processors<3 then node 722 elseif Processors>=3 then node 723 else krylovshur

class = arnoldi
class = krylovshur
class = krylovshur
class = arnoldi
class = arnoldi
class = krylovshur
class = krylovshur

if Processors<72 then node 724 elseif Processors>=72 then node 725 else arnoldi

if Spectrum=1 then node 726 elseif Spectrum=3 then node 727 else arnoldi
class = arnoldi

if Size<22204 then node 728 elseif Size>=22204 then node 729 else arnoldi

if Size<22204 then node 730 elseif Size>=22204 then node 731 else krylovshur

class = krylovshur
class = arnoldi
class = arnoldi
class = krylovshur
class = gd

if Processors<3 then node 732 elseif Processors>=3 then node 733 else jd

class = lanczos
class = krylovshur
class = krylovshur
class = lanczos
class = lanczos
class = krylovshur
class = krylovshur
class = lanczos
class = arnoldi

if Size<696 then node 734 elseif Size>=696 then node 735 else arnoldi
class = lanczos
class = arnoldi
class = lanczos
if Processors < 36 then node 736 elseif Processors >= 36 then node 737 else arnoldi
    krylovshur
class = krylovshur
if NoOfEigvalues < 1.5 then node 740 elseif NoOfEigvalues >= 1.5 then node 741 else krylovshur
if Processors < 144 then node 742 elseif Processors >= 144 then node 743 else krylovshur
class = arnoldi
class = krylovshur
if Rank = 1 then node 744 elseif Rank = 2 then node 745 elseif lanczos
if Rank = 1 then node 746 elseif Rank = 2 then node 747 elseif krylovshur
if Rank = 1 then node 748 elseif Rank = 2 then node 749 elseif krylovshur
if Rank = 1 then node 750 elseif Rank = 2 then node 751 elseif lanczos
if Rank = 1 then node 752 elseif Rank = 2 then node 753 elseif arnoldi
if Spectrum = 1 then node 754 elseif Spectrum = 3 then node 755 elseif arnoldi
class = lanczos
class = krylovshur
class = arnoldi
class = arnoldi
class = krylovshur
class = krylovshur
class = arnoldi
class = krylovshur
if Processors < 18 then node 762 elseif Processors >= 18 then node 763 elseif arnoldi
class = arnoldi
class = krylovshur
class = arnoldi
if Tolerance < 5.0005e−05 then node 764 else Tolerance ≥ 5.0005e−05 then node 765 else arnoldi

class = krylovshur

class = lanczos

class = krylovshur

class = arnoldi

class = arnoldi

class = krylovshur

class = arnoldi

if Processors < 144 then node 766 else if Processors ≥ 144 then node 767 else krylovshur

if Rank = 1 then node 768 else if Rank = 2 then node 769 else lanczos

class = arnoldi

if Processors < 36 then node 770 else if Processors ≥ 36 then node 771 else krylovshur

if Processors < 36 then node 772 else if Processors ≥ 36 then node 773 else lanczos

class = arnoldi

class = krylovshur

class = krylovshur

class = arnoldi

class = arnoldi

class = krylovshur

class = krylovshur

class = arnoldi

class = krylovshur

class = arnoldi

class = krylovshur

class = jd

class = gd

if Tolerance < 5.05e−09 then node 774 else if Tolerance ≥ 5.05e−09 then node 775 else arnoldi
if Processors < 144 then node 776 elseif Processors >= 144 then node 777 else
arnoldi

class = arnoldi

if Spectrum = 1 then node 778 elseif Spectrum = 3 then node 779 else krylovschur

class = arnoldi

if Processors < 144 then node 780 elseif Processors >= 144 then node 781 else
krylovschur

class = lanczos

class = krylovschur

if NoOfEigvalues < 1.5 then node 782 elseif NoOfEigvalues >= 1.5 then node 783 else
krylovschur

class = arnoldi

if Processors < 10 then node 784 elseif Processors >= 10 then node 785 else arnoldi

if Processors < 72 then node 786 elseif Processors >= 72 then node 787 else
krylovschur

class = krylovschur

class = arnoldi

class = krylovschur

class = arnoldi

class = lanczos

class = power

class = krylovschur

class = arnoldi

class = power

class = krylovschur

class = krylovschur

class = arnoldi

class = krylovschur

class = arnoldi

class = krylovschur

class = arnoldi

class = krylovschur

class = arnoldi
if Processors < 72 then node 788 elseif Processors >= 72 then node 789 else krylovshur

class = lanczos
class = lanczos
class = krylovshur
class = lanczos
class = krylovshur
class = krylovshur
class = lanczos
class = krylovshur
class = arnoldi
class = krylovshur
class = arnoldi
class = arnoldi
class = krylovshur
class = krylovshur
class = lanczos
class = arnoldi
class = krylovshur
class = lanczos
class = arnoldi
class = krylovshur
class = lanczos
class = arnoldi
class = krylovshur
class = lanczos
class = arnoldi