Generating Customized Sparse Eigenvalue Solutions with Lighthouse

Ramya Nair*, Sa-Lin Bernstein†, Elizabeth Jessup* and Boyana Norris‡
*Department of Computer Science, University of Colorado Boulder, Boulder, CO, USA
Email: ramya.nair@colorado.edu, elizabeth.jessup@colorado.edu
†Computation Institute, University of Chicago and Argonne National Laboratory, Chicago, IL, USA
Email: salin@anl.gov
‡Department of Computer and Information Science, University of Oregon, Eugene, OR, USA
Email: norris@cs.uoregon.edu

Abstract—**Sparse eigenvalue problems arise in many areas of scientific computing. A variety of high-performance numerical software packages including many different eigensolvers are available to solve such problems. The two main challenges are finding the routines that can correctly solve the problem and implementing the desired solution accurately and efficiently using the appropriate software package. In this paper, we describe an approach that addresses these issues by intelligently identifying the sparse eigensolver that is likely to perform the best for given input characteristics and by generating a code template that uses that solver. The results are delivered to users through Lighthouse, a novel interface and search platform for users seeking high-performance solutions to linear algebra problems. This paper describes the development of the approach with a focus on the analysis of sparse eigensolvers in SLEPc and their integration into Lighthouse.**

Keywords–expert systems; sparse eigensolvers; machine learning.

I. INTRODUCTION

Sparse eigenvalue problems are pervasive in scientific computing, engineering, and optimization-based machine learning methods. Such problems appear in various fields ranging from subatomic particle theories in quantum physics [1] to structural engineering [2]. In recent years, they are also increasingly used in search engines [3] and social networks [4]. Scientists and engineers typically use high-performance libraries that were developed over time by teams of numerical linear algebra experts. However, solving sparse eigenvalue problems accurately and efficiently with these packages normally takes significant effort and requires knowledge in applied mathematics, computational methods, and the problem domain. One of the commonly known example of sparse eigenvalue problem is the pagerank algorithm used in Google's search [3]. A modified version of power method was used to solve the search problem.

One of the common challenges faced by the developers is the lack of knowledge of different iterative methods for solving eigenvalue problems. Some methods or algorithms may converge much faster than others for the particular problem at hand. Hence, an inappropriate solver selection or configuration may lead to an unwanted result such as a high residual or no convergence. Another challenge lies in selecting a suitable library and identifying the most appropriate routines for a given problem. These processes, generally, depend 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 programming language. Optimizing the performance of an implementation is a big challenge for many developers because it requires a solid understanding of the selected software package framework, numerical computation, compilers, and computer architecture.

Lighthouse [5] is an innovative framework that connects linear algebra software resources with code implementation and optimization. Lighthouse is an ongoing project with ever-increasing content and functionality. This paper focuses on the addition of eigensolvers from the Scalable Library for Eigenvalue Problem Computations (SLEPc) [6] to Lighthouse and is organized as follows. Section II provides the related work. Section III discusses our approach for developing the Lighthouse taxonomy. Section IV presents the details of the integration of SLEPc with Lighthouse to identify and deliver the best sparse eigensolvers to users. Section V summarizes the conclusions and outlines future work.

II. RELATED WORK

A number of existing taxonomies attempt to address the problem of finding and using high-performance numerical software. Perhaps the oldest one (starting in 1985) is the Netlib Mathematical Software Repository [7], which contains freely available software, documents, and databases pertaining to numerical computing including eigensolvers. The information is organized as lists of packages or routines, with or without accompanying documentation. The newer Linear Algebra Software Survey [8] contains over sixty items categorized as support routines, dense direct solvers, sparse direct solvers, preconditioners, sparse iterative solvers, and sparse eigenvalue solvers together with a checklist specifying problem types for each entry. NIST's Guide to Available Mathematical Software (GAMS) [9] includes a wider range of basic linear algebra software along with software for a variety of other numerical applications. While the Linear Algebra Software Survey is a linear list without advanced search capabilities, GAMS allows search by problem solved, package name, module name, or text in the brief module abstract. An earlier Java-based client called HotGAMS [10] allowed an interactive search of the GAMS repository, but is no longer available. Both the Survey and GAMS index into Netlib for software downloads. It is also possible to browse and search Netlib directly.

With existing taxonomies or general-purpose search engines, the user must manually explore the many available packages and learn enough about each of them to be able to make a good choice. For full understanding, the user may also need to read significant portions of the documentation for each candidate software package. After selecting a library that can solve his or her target problem, the user typically spends considerable time learning how to use it correctly and efficiently before they can encode their solution.

Even after rigorous comparative analysis of the methods and careful selection of software packages, the selected algorithm may or may not work for the problem of interest depending on various factors. When using high-quality software, an unsatisfactory solution may still result for certain inputs. 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.

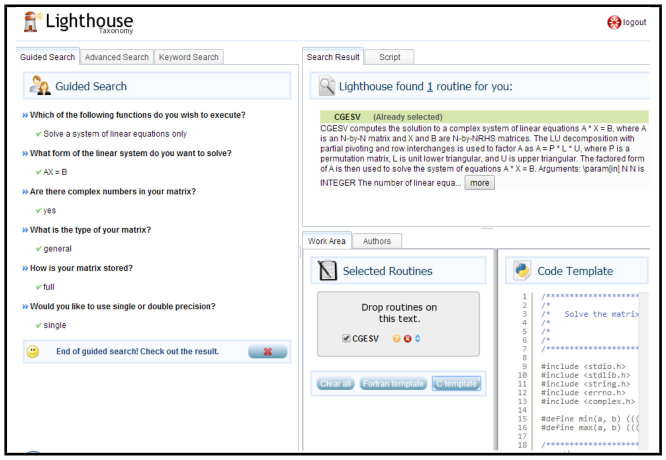
III. PROPOSED APPROACH

Lighthouse is an open source web-based expert system that matches a user's functional and performance needs with available high-performance linear algebra software. The Lighthouse taxonomy provides a classification of existing linear algebra libraries that currently includes Linear Algebra PACKage (LAPACK) [11], Portable, Extensible Toolkit for Scientific Computation (PETSc) [12], and SLEPc. Built with the Django [13] web application framework, the Lighthouse user interface (UI) is a user-centered design, offering efficient search capabilities that accommodate users with various levels of experience in numerical linear algebra. Figure 1 illustrates the Lighthouse Guided Search for LAPACK linear solver routines. The Guided Search is designed to lead users through increasingly refined subroutine searches until the desired result is attained. The Advanced Search, on the other hand, is recommended for users who are familiar with the library. All users can benefit from the Keyword Search, which allows for subroutine search via an input keyword or phrase.

In addition to the search feature, Lighthouse creates code templates in FORTRAN 90 and C containing working programs that declare and initialize required data structures and call selected subroutines. Users can download and modify the templates to meet their particular project needs. Moreover, 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) [14] compiler. BTO produces highly tuned C implementations based on high-level MATLAB-like input specification of the computation.

The Lighthouse taxonomy is continuously expanding, and SLEPc is one of our current development focuses. SLEPc is based on the PETSc package [15] [12], which provides a comprehensive set of data structures and algorithms for the parallel solution of problems modeled with nonlinear partial differential equations.

To determine what eigensolver is the most appropriate for a given problem, we have analyzed SLEPc solvers by applying machine learning techniques to a large set of different problems to find those that are most likely to yield the best performance for a given set of matrix features. The next section describes the process of integrating SLEPc with Lighthouse.



Guided Search

» Which of the following functions do you wish to execute?
✓ Solve a system of linear equations only
» What form of the linear system do you want to solve?
✓ AX = B
» Are there complex numbers in your matrix?
✓ yes
» What is the type of your matrix?
✓ general
» How is your matrix stored?
✓ full
» Would you like to use single or double precision?
✓ single

End of guided search! Check out the result.

Lighthouse found 1 routine for you:

CGESV (Already selected)
CGESV computes the solution to a complex system of linear equations A * X = B, where A is an N-by-N matrix and X and B are N-by-NRHS matrices. The LU decomposition with partial pivoting and row interchanges is used to factor A as A = P * L * U, where P is a permutation matrix, L is unit lower triangular, and U is upper triangular. The factored form of A is then used to solve the system of equations A * X = B. Arguments: \param[in] N N is INTEGER The number of linear equa... more

Drop routines on this text.
✓ CGESV

```
/*******************
/*
/*   Solve the matrix
/*
/*
/*
/*******************

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <complex.h>

#define min(a, b) (((
#define max(a, b) (((

/*******************
```

Figure 1. Lighthouse Guided Search for LAPACK linear solver routines.

TABLE I. INPUT FEATURE SET.

Property tested	**Range of input parameters tested**
Matrix order	112 x 112 to 262,111 x 262,111
Matrix type	Real, Complex
Matrix data	Binary, Double
Matrix characteristics	Hermitian, Non-Hermitian
Number of eigenvalues	1, 2, 5, 10
Portion of spectrum	Largest magnitude, Smallest magnitude, Largest real, Smallest real, Largest imaginary, Smallest imaginary
Tolerance	1.00E-04, 1.00E-08, 1.00E-10
Number of processors	1, 2, 4, 8, 12, 24, 48, 96, 192

TABLE II. SLEPC SOLVERS TESTED.

SLEPc solvers	power, subspace, arnoldi, lanczos, krylovschur, generalized davidson, jacobi davidson

IV. SLEPC - LIGHTHOUSE INTEGRATION

We experimented with different eigenvalue problems and available SLEPc solvers to determine the performance of each solver. First, we considered known functional properties of algorithms to narrow down to the solvers that work best for each problem examined (e.g., some methods are only suitable for solving symmetric problems). We then used machine learning techniques to intelligently predict the solvers that will work best for a given set of input features. The experimental results served as training data to the prediction algorithm, which identifies the sparse eigensolvers included in Lighthouse for a given problem.

A. Experiments

To run the experiments, we first obtained matrices from Matrix Market [16] and the Florida Sparse Matrix Collection [17] that cover the problem domains of interest.

Table I shows the range of input parameters tested. For each case of these input parameters, we obtained the performance and result accuracy for different SLEPc solvers. Table II shows the SLEPc solvers used for these experiments. The resultant data set consists of more than 29,000 data points.

B. Training and Prediction

To begin, we experimentally determined the solvers that work best for every case tested, after which, we used machine learning techniques to make intelligent decisions about the best known solvers for any untested case. The predicted results were then verified using standard validation techniques. The following subsections describe these steps in detail.

1) Finding the best solver - Training data setup: The analysis to identify the best suited eigensolvers can be split into two steps: elimination and selection. Output characteristics of interest for these two steps are the number and selection of converged eigenpairs, time taken, and residual.

In the elimination step, we first remove the solvers with resultant characteristics completely outside the expected value range. For this, we count the number of eigenvalues that converged with a residual less than the given residual tolerance. If the count obtained is less than the desired number of eigenvalues, the solver is removed from the list.

For the selection step, all of the eigensolvers remaining after the described culling are reasonable choices, but the fastest of them is selected as the best fit for the given problem specifications. Additionally, to avoid losing other efficient solvers, we select from the remaining list those that take at most 10% more time than the best fit eigensolver.

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.

Non-convergence case: After experimenting with very large upper limits, we set the maximum iteration limit to 1,000 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 5,000. The experiments established that most solutions converge significantly fewer itrations. If the solution does not converge within the specified limits, it is labelled as non-convergent.

In such non-convergent cases, there is still a possibility that a solver converges but not to the exact expected number of eigenvalues or residual tolerance. Solution methods that show some signs of convergence, even though not to the desired values, are retained for the training data. Hence, such input cases have the information of the partially converged solvers along with non-convergent label.

2) Intelligent solver selection - Prediction: The previous step produces a reduced set of data which gives the best solvers for each tested case. We apply machine learning techniques to make intelligent predictions for other problems using the data as a training set.

Decision tree induction [18] is a 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, the matrix properties and expected output parameters to form a pattern that can best fit the training data. The final prediction model is a *classification tree*, where every inner node is an attribute to be selected, every branch from the node is a selection made, and the leaf nodes lead to the predicted result. For our case, we want the data to be classified into the eigensolvers available in SLEPc.

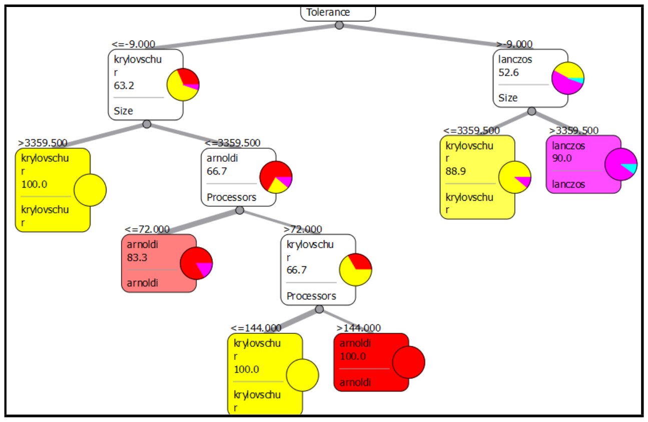


Figure 2. Part of the generated classification tree using ORANGE.

From the previous step, for every input case tested, we have one or more best solvers. To accommodate multiple solver recommendations, apart from the input parameters, we added a performance index feature to the dataset. For every input case, the performance index is given an integer value of one for the fastest solver and is incremented by one for remaining solvers with increasing time taken. Using the expanded feature set as training data, the obtained decision tree learns the order of the best performing eigenvalue solvers as well. The generated tree can be used to suggest multiple eigensolver classes by simply ignoring the performance index and taking into account all other feature criteria in the query.

We employed two applications to obtain the classification tree from the training data. We first used MATLAB's *ClassificationTree.fit* functionality [19]. We also used the application Orange [20], which is an open-source data mining tool for different learning operations, such as classification, evaluation and prediction. In particular, Orange has good visualization options that make viewing large trees much easier than with other applications like MATLAB or Weka [21].

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. A portion of the classification tree generated from Orange is shown in Figure 2. 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 do not converge to a solution for any of the eigensolvers available in SLEPc. Such leaf nodes have the value set to "No Convergence".

3) Validation: We employed several methods to validate the results. For the decision tree in MATLAB, cross-validation results were checked using the MATLAB `cvLoss()` function. The decision tree cross-validation classification error (loss) for the training data set was obtained to be 0.1450 which implies a 14.5 percent error in prediction. For the decision tree created using Orange, 10-fold cross-validation was applied,

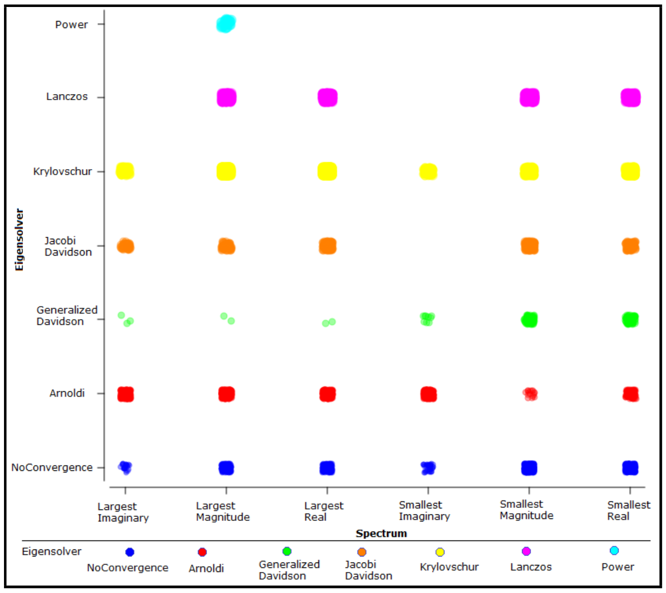
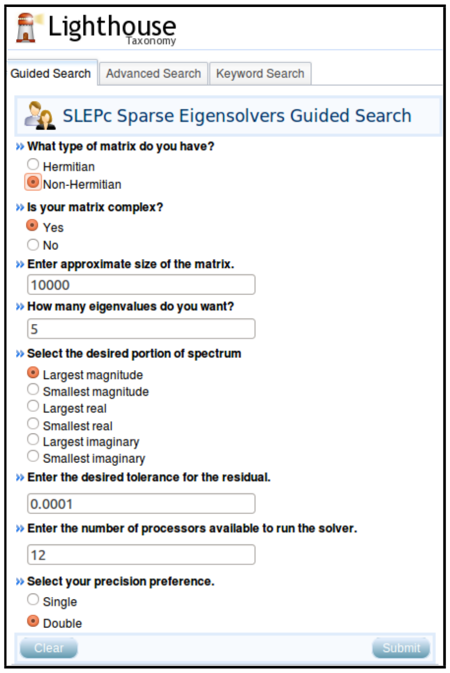


Figure 3. Scatter plot analyzing eigensolver vs portion of spectrum.


Lighthouse
Taxonomy

Guided Search | Advanced Search | Keyword Search

SLEPc Sparse Eigensolvers Guided Search

» What type of matrix do you have?
Hermitian [✓ Non-Hermitian]

» Is your matrix complex?
[✓ Yes] No

» Enter approximate size of the matrix.
10000

» How many eigenvalues do you want?
5

» Select the desired portion of spectrum
[✓ Largest magnitude] Smallest magnitude Largest real Smallest real Largest imaginary Smallest imaginary

» Enter the desired tolerance for the residual.
0.0001

» Enter the number of processors available to run the solver.
12

» Select your precision preference.
Single [✓ Double]

Clear Submit

Figure 4. Guided search UI in Lighthouse for sparse eigensolver routines

and we obtained classification accuracy of 0.8639. Close to the results from MATLAB, the latter result implies a 13.61 percent error in prediction.

The validation results obtained using the above methods take into account the order of the best performing solvers. Lighthouse implementation suggests all the favorable solvers irrespective of the order. Hence, to get more accurate validation results with respect to Lighthouse, we employed a different technique. We used the generated tree to predict the results for each input case tested, varying the performance index. As a result, we formed a favorable solver subset for every input case, each predicted solver in the subset corresponds to a different performance index. If the predicted favorable solver subset is same as the expected subset (in training data), irrespective of the performance index, we considered it to be accurately predicted. This test gives an error of 7.99%, which implies 92.01% input cases were accurately predicted.

4) Result analysis: In this section we describe some result analysis with respect to the experiments conducted. Figure 3 shows a scatter plot of the eigensolver versus a portion of spectrum. Note that jacobi davidson (jd) is not a preferred solver for any smallest imaginary case and that generalized davidson (gd) is less likely to be selected as the most efficient solver for the largest portion of the spectrum. Similarly, plotting eigensolvers versus binary and non-binary matrices (where "binary" refers to a matrix with all non-zero values as one), we found that lanczos solver, which predominantly worked for Hermitian non-binary matrices, does not work at all for Hermitian binary matrices.

Plotting these parameters in graph form helps us infer some feature characteristics in two dimensions, whereas the decision tree captures all such results in higher dimensions (for every feature tested).

C. User Interfaces

Once the collection of the data required for integrating SLEPc with Lighthouse is complete, we constructed a MySQL database through Django. The database schema is derived from the information on the decision tree.

We converted the classification tree information into a MySQL datatable using a depth first search algorithm [22]. Every path from the root node to the leaf node forms a single row in the datatable. The columns of the datatable constitute 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 residual tolerance and the number of eigenvalues. The generated decision tree consists of 395 leaf nodes. Thus, the resulting datatable comprises 395 distinct rows with unique features. The user selects the desired features via the Lighthouse-SLEPc interface and the respective best solvers are fetched from the database.

Like the LAPACK UI in Lighthouse, the SLEPc UI also provides Guided Search. The questions correspond to the features in the generated decision tree. Each question is a fundamental query to the database that directs the search toward the most appropriate result. Upon the user's answering all the questions, Lighthouse suggests the best known eigensolvers, and the user may choose to generate a respective code template in FORTRAN 90 or C. Figure 4 illustrates the Guided Search UI in Lighthouse for sparse eigensolver routines using the SLEPc package. Note that the interface consists of simple questions that lead the user to the desired SLEPc routines.

The Advanced Search implementation contains questions similar to Guided Search, but it delivers all the SLEPc eigensolver routines that are compatible with the user-specified problem (irrespective 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 also includes some questions requiring extended eigensolver knowledge.

The Keyword Search is also provided so that users can directly search for the routine by name and generate the respective code template.

V. CONCLUSION AND FUTURE WORK

In this paper, 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 the machine learning techniques will offer more accurate and efficient solutions and relieve users from having to research the documents of every eigensolver routine available. 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 work, we will continue to integrate more SLEPc functionality (e.g., singular value decompositions) into Lighthouse and enable all of the described search features. 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, thereby further accelerating scientific development and discoveries.

ACKNOWLEDGMENT

This work is supported by the National Science Foundation (NSF) grants CCF-1219089 and CCF-091474. We used the Janus supercomputer, which is supported by the NSF award 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. Janus is operated by the University of Colorado Boulder.

REFERENCES

[1] R. H. Landau, J. Paez, and C. C. Bordeianu, A survey of computational physics: introductory computational science. Princeton University Press, 2011.

[2] K.-J. Bathe and E. L. Wilson, “Solution methods for eigenvalue problems in structural mechanics,” International Journal for Numerical Methods in Engineering, vol. 6, no. 2, 1973, pp. 213–226.

[3] K. Bryan and T. Leise, “The $25,000,000,000 eigenvector: The linear algebra behind Google,” Siam Review, vol. 48, no. 3, 2006, pp. 569–581.

[4] X. Ying and X. Wu, “Randomizing social networks: a spectrum preserving approach.” in Proceedings of the 8th SIAM Conference on Data Mining (SDM08), vol. 8. SIAM, 2008, pp. 739–750.

[5] Lighthouse project. https://code.google.com/p/lighthouse-taxonomy/. [retrieved: May, 2014]

[6] V. Hernandez, J. E. Roman, and V. Vidal, “SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems,” ACM Trans. Math. Software, vol. 31, no. 3, 2005, pp. 351–362.

[7] Basic Linear Algebra Subprograms (BLAS). [Online]. Available: http://www.netlib.org/blas [retrieved: May, 2014]

[8] J. Dongarra. Freely available software for linear algebra on the web. http://www.netlib.org/utk/people/JackDongarra/la-sw.html. [retrieved: Jan, 2014]

[9] NIST. Guide to Available Mathematical Software (GAMS). http://gams.nist.gov. [retrieved: Jan, 2014]

[10] ——. HotGAMS: Java interface to Guide to Available Mathematical Software. [retrieved: Jan, 2008]

[11] LAPACK - Linear Algebra PACKage. [Online]. Available: http://www.netlib.org/lapack/ [retrieved: May, 2014]

[12] PETSc - Portable, Extensible Toolkit for Scientific Computation. [Online]. Available: http://www.mcs.anl.gov/petsc/ [retrieved: May, 2013]

[13] Django. [Online]. Available: https://www.djangoproject.com/ [retrieved: May, 2014]

[14] J. G. Siek, I. Karlin, and E. R. Jessup, “Build to order linear algebra kernels,” Workshop on Performance Optimization of High-level Languages and Libraries. POHLL, April 2008, pp. 1–8.

[15] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, “Efficient management of parallelism in object oriented numerical software libraries,” in Modern Software Tools in Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds. Birkhäuser Press, 1997, pp. 163–202.

[16] Matrix Market. [Online]. Available: http://math.nist.gov/MatrixMarket/ [retrieved: June, 2013]

[17] The University of Florida Sparse Matrix Collection. [Online]. Available: http://www.cise.ufl.edu/research/sparse/matrices/ [retrieved: Feb, 2014]

[18] J. Han, M. Kamber, and J. Pei. Data mining concepts and techniques, third edition. Waltham, Mass. (2012)

[19] MATLAB, Release R2013b. Natick, Massachusetts: The MathWorks Inc., 2013.

[20] Orange. [Online]. Available: http://orange.biolab.si/ [retrieved: May, 2014]

[21] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA data mining software: An update,” SIGKDD Explor. Newsl., vol. 11, no. 1, Nov. 2009, pp. 10–18. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278

[22] S. S. Skiena, The Algorithm Design Manual. New York, NY, USA: Springer-Verlag New York, Inc., 1998.

[23] V. Hernandez, J. E. Roman, A. Tomas, and V. Vidal, “A survey of software for sparse eigenvalue problems,” Universitat Politècnica de València, Tech. Rep. STR-6, [retrieved: May, 2013]. [Online]. Available: http://www.grycap.upv.es/slepc

[24] C. Campos, J. E. Roman, E. Romero, and A. Tomas, “SLEPc users manual,” D. Sistemes Informàtics i Computació, Universitat Politècnica de València, Tech. Rep. DSIC-II/24/02 - Revision 3.3, 2012.

[25] V. Hernandez, J. E. Roman, A. Tomas, and V. Vidal, “Single vector iteration methods in SLEPc,” Universitat Politècnica de València, Tech. Rep. STR-2, [retrieved: May, 2013]. [Online]. Available: http://www.grycap.upv.es/slepc

[26] ——, “Arnoldi methods in SLEPc,” Universitat Politècnica de València, Tech. Rep. STR-4, [retrieved: May, 2013]. [Online]. Available: http://www.grycap.upv.es/slepc

[27] ——, “Lanczos methods in SLEPc,” Universitat Politècnica de València, Tech. Rep. STR-5, [retrieved: May, 2013]. [Online]. Available: http://www.grycap.upv.es/slepc

[28] ——, “Krylov-schur methods in SLEPc,” Universitat Politècnica de València, Tech. Rep. STR-7, [retrieved: May, 2013]. [Online]. Available: http://www.grycap.upv.es/slepc

[29] ——, “Davidson type subspace expansions for the linear eigenvalue problem,” Universitat Politècnica de València, Tech. Rep. STR-10, [retrieved: May, 2013]. [Online]. Available: http://www.grycap.upv.es/slepc

[30] Y. Saad, Numerical Methods for Large Eigenvalue Problems. Halsted Press, NY, 1992.

[31] L. N. Trefethen and D. Bau, Numerical Linear Algebra. SIAM, 1997.

[32] N. S. Foundation and D. of Energy. BLAS. http://www.netlib.org/blas/. [Online]. Available: http://www.netlib.org/blas/ [retrieved: May, 2013]

[33] R. Nair, “Customized sparse eigenvalue solutions in Lighthouse,” Master’s thesis, University of Colorado Boulder, May 2014.