About Me

Username and Contact Information

Name: Ramit Pahwa

University : Indian Institute of Technology (IIT), Kharagpur (http://iitkgp.ac.in)

Email : ramitpahwa123@iitkgp.ac.in | ramitpahwa123@gmail.com

Github Username : Ramit-Pahwa (https://github.com/Ramit-Pahwa)

Personal Background

I am a fourth-year undergraduate pursuing Mathematics and Computing, IIT Kharagpur, India.

My primary workstation has macOS High-Sierra and I use Atom as my preferred text editor.I have programmed in C and C++ for the past 5 years and in python for the past 2 years or so and am proficient in all three of them. I am deeply passionate about Machine Learning and have been actively researching on Deep Learning Architectures.

I am also familiar with git for version control and have successfully completed projects/internship with large teams.

My educational background is sufficient for the project, as I have been exposed to various area's of mathematics through my curriculum which has imparted me a strong foundation.I am an avid reader of fiction and true fan of FC Barcelona.

Previous Experience

My first encounter with the Algebra was during my 3rd year of college when I took an Introduction course on Linear Algebra. As a part of the curriculum, I have been constantly doing mathematical modeling in C++ and Python.(Links to repository below). I have experience in working in a team through internships I did at the University of Alberta, where I was awarded best student research paper, as well as working in developing production level Machine Learning/ Deep Learning solutions at Myntra.

Relevant Courses

Mathematics

- Operation Research (Theory and Lab) LINK (https://github.com/Ramit-Pahwa/Operations-Research)
- Computational Linear Algebra (ongoing)
- Advanced Numerical Techniques (Theory and Lab) LINK (https://github.com/Ramit-Pahwa/Advance-Numerical-Techniques)
- Linear Algebra

Computer Science

- Algorithms And Data Structure (Theory and Lab)
- Object-Oriented Systems Design (Theory and Lab)
- Computer Organization (Theory and Lab)
- Machine Learning and Deep Learning (Theory and Project)

Answers of some questions

`1. What do you want to have completed by the end of the program?`

By the end of the program, I want to add support for solving generalized eigenvalue problem efficiently and remove the dependency on ARPACK.

`2. Who's interested in the work, and how will it benefit them?`

Many problems in applied sciences are posed as eigenvalue problems an efficient solver for the large sparse matrix using Arnoldi method adds value to Julia as a package.

`3. What are the potential hurdles you might encounter, and how can you resolve them?`

Proposal for the project explains the above.

`4. What other time commitments, such as summer courses, other jobs, planned vacations, etc., will you have over the summer?`

I expect to work full time on the project that is 40 or more hours a week. I have no other commitments in the summer as of now.

Experience with Julia

I have been using Julia for last one and half month. In terms of functionality, I like Julia because of its **multiple dispatch** feature as it lets me overload operators with a lot of ease than other programming languages.

But the what intrigues me the most is the ability to develop new packages without going into the intricacies of low-level language and compromising with speed.Plus its similarity with python and Matlab make it easy to understand and reproduce.

The most amazing feature of Julia is its parallelism, parallelize directly from the command line by calling the desired script with a given number of cores.Additionally, it also has the ability to assign a task to different thread directly from the code.

The Project

The Problem and Motivations

The objective of the project is to add iterative methods for solving the (generalized) eigenvalue problem `Ax = λBx`.The goal of this project is to implement the Arnoldi or Lanczos method as a solver for the eigenvalue problem and to compare its performance with the `eigs` function.

Many practical problems in Computer Science and Applied Mathematics are posed as eigenvalue problems, Hence, an efficient and quick solver for the problem is highly desirable. A typical source of large-scale eigenproblems is through a discrete form of a continuous problem. The resulting finite dimensional problems become large due to accuracy requirements and spatial dimensionality.To my understanding, a correct approach would be to write a separate package **IRAM.jl** dealing with the generalized eigenvalue problem through implementation of Arnoldi method which is typically suitable for computation of a small number of eigenvalues in cases where the matrix is large and sparse.

The Plan

I propose to implement required functionality in a package IRAM.jl tacking the problem in parts.First looking at the problem `Ax = λx`, then extending the implementation to the generalized eigenvalue problem `Ax = λBx` I plan to implement a 5 step procedure to implement the solver for generalized eigenvalue problem and document my code as I progress as well as create a blog of the regular

1. Add efficient Implementation of Arnoldi factorization-base module.
2. Add Implicitly restarted Arnoldi method using efficient implementation of shifted QR strategy.
3. Handle specific target eigenvalues (such as largest/smallest real/imaginary part or maximum absolute magnitude) along with various stopping criterion and deflation,
4. Locking and Purging of ritz value through Orthogonal Deflating Transformation
5. Implementation of the solver for general problem `Ax = λBx` where `B` is symmetric positive definite (happens in FEM a lot) or potentially singular. **Implementation of Lancoz,block methods [post GSOC period]**

I also aim to publish the work done during this period under the guidance of my mentor and continue to contribute and build the package beyond Summer of Code period.

Mathematical Formulation

First of all, we need to understand the mathematical formulation of Eigen problem before we get into the implementation details.

Definitions

Krylov subspace: In linear algebra, the order-r Krylov subspace generated by an n-by-n matrix A and a vector b of dimension n is the linear subspace spanned by the images of b under the first r powers of A (starting from `A^0=I A^0=I`), that is,

$$K_r(A, b) = span\{b, Ab, A^2b, A^3b, \ldots, A^{r-1}b\}$$

Orthonormal basis: an orthonormal basis for an inner product space V with finite dimension is a basis for V whose vectors are orthonormal, that is, they are all unit vectors and orthogonal to each other.

In the process, we obtain an upper Hessenberg matrix, H, of dimension m whose eigenvalues are estimations of the dominant eigenvalues of matrix `A`. The process to build the matrix `H` is based on the following algorithm

1.Arnoldi Algorithm

1. **START :** Choose an initial unitary vector, `x1`. This vector can be chosen by the user (random or determined) or from the restart file. Choose the Krylov subspace dimension, `m`. Choose the desired

number of converged eigenvalues, **nev** .

2. Set

$$k = 1.w = Ax_1, h_{11} = x_1^T w, f_1 = w - x_1 h_{11}$$

3. **Arnoldi Loop** . Calculate

$$for\ j = k, k+1, \ldots m-1\ do$$
$$h_{j+1,j} = \|f_j\|$$
$$x_{j+1} = \frac{f_j}{h_{j+1,j}}$$
$$w = Ax_{j+1}$$
$$h_{i,j+1} = x_i^T w,\ i = 1, 2, 3 \ldots, j$$
$$f_{j+1} = w - \sum_{i=1}^{j} x_i h_{i,j+1}$$
$$End\ do$$

Normally, the value of `m` is quite lower than the dimension of matrix `A, n`, and if we only make one iteration of the algorithm, the eigenvalues of matrix `H` are not good approximations of the dominant eigenvalues of matrix `A` unless we choose the Krylov subspace dimension, `m`, very large. Thus, it is necessary to reinitialize the process. To do this, we compute the real Shur form of matrix `H`

$$T = Z_T H Z$$

and calculate

$$X = XZ$$

The first column of `X`, normalized to one is taken as reinitialization vector, `x1` and the algorithm returns to step 2, if the process is not converged. If the `kth` eigenvalue is converged, we can make use of a deflated process (Saad, 1992), set `k = k +1`, and return to step 3.

When `nev` eigenvalues of matrix `H` converge, we obtain as results of the process these eigenvalues as approximations for the `nev` dominant eigenvalues of matrix `A`, and the corresponding eigenvectors are linear combinations of the first `nev` columns of matrix `X`. Normally m is chosen higher than `2nev+1`. **below is an implemented version of the Arnoldi iteration.**

```
# function for arnoldi method

function arnoldi_procedure(A,m::Int64)
    # Procedure to generate orthogonal basis of the Krylov subspace
    n = size(A,1)
    Q = Matrix{Float64}(n,m+1)
    h = zeros(m+1,m)
    v = rand(n)
    w = Vector{Float64}(n)
    b=view(Q,:,1)
    copy!(b,v)
    normalize!(b)
    # fview(q,1)=normalize(V)
    for j = 1:m
        A_mul_B!(w,A,view(Q,:,j))

        # Gram-Schmidt Orthogonalization
        for i = 1:j
            q1 = view(Q,:,i)
            # h1 = view(h,i,j)
            h[i,j] = dot(w,q1)
            # zero dimensional subarray
            w.-=q1.*h[i,j]
        end
        #h1 = view(h,j+1,j)
        #copy!(h1,norm(w,1))
        h[j+1,j] = norm(w,1)
        if h[j+1,j] == 0 # Found an Invariant Subspace
            return Q,h
        end
        q1=view(Q,:,j+1)
        copy!(q1,w./h[j+1,j])
    end
    return Q,h
end
```

The main computational cost of the algorithm is the product matrix-vector. This process has a slow convergence and it needs a large number of iterations. An improvement of the basic algorithm, which reduces the number of matrix-vector products, is the Sorensen's IRA method.

2.Implicitly Restarted Arnoldi Method (IRAM)

IRA method is scheduled in the following algorithm
STEP 1 : Perform the m-step Basic Arnoldi method to obtain `X, H and f_m`
STEP 2 : Do until convergence

1. Compute the eigenvalues of H, θ_1; ... θ_m, and their corresponding eigenvectors, s_1,.. s_m and select p undesired eigenvalues θ_{m-p+1},. . .,θ_m (p must be such that $nev < m - p$).Note that the eigenvalues θ_i are approximations to the eigenvalues of matrix A.

2. Set $Q = (I)_{mxm}$
3. for j=1 ,2 ... p do
   - Compute the QR Factorization
   - $Q_j R = H - \theta_{m-(p-1-j)} I$
   - Compute
   $$H = Q_j^T H Q_j \ \ Q = QQ_j$$
4. End Do
   - Set $X = XQ$
   - Compute $f_k = x_{k+1} h_{k+1,k} + f_m q_{mk}$ where $k = m - p$
   - Use the Arnoldi loop,of the the basic arnoldi algorithm ,with $k = m - p$ to obtain $X, H \ \ and \ \ f_m$
   - Check the convergence criterion
   $$\|f_m\| |e_m^T| \leq max(\gamma_m \|H\|, tol|\theta_i|) \ \ i = 1, \ldots, nev.$$
   where $e_m^T = (0, 0, 0, .., 1)_{1xm}, \ \ \gamma_m$ is machine precision and tol is prefixes tolerance.
   End do

STEP 3 : When the matrix X is converged, we store the first column, $x1$, in the restart file Once the process above has converged, to compute the dominant eigenvalues of A and the corresponding eigenvectors, the algorithm finishes with the following steps:

1. Compute the partial Shur form
$$HQ_{nev} = Q_{nev} R_{nev} \ \ where \ \ Q_{nev} \epsilon >= R^{mxnev} \ \ R_{nev} \epsilon >= R^{nevxnev}$$
2. Store in the first $nev0$ columns of X:
$$x_i = X(Q_{nev}), \quad i = 1, .., nev$$
3. Compute the eigenvalues decomposition
$$R_{nev} S_{nev} = S_{newnev}$$
where S_{newnev} is diagonal matrix whose elements are the obtained approximations to the dominant eigenvalues of A
4. The Corresponding eigenvectors are given by:
$$X_{nev} = X_{nev} S_{nev}$$
where X_{nev} are the matrix whose columns are the vectors obtained above in 2

This method takes advantage of the shifted QR iterations to accelerate the process. These iterations have a very low computational cost because of the size of the matrix H is small and they reduce the number of large matrix-vector products, Ax, necessary for the convergence of the whole process.A good resource to understand subtlety in implementation is [1],[2].

```
#QR Factorization step

A = [3.0 -6.0; 4.0 -8.0; 0.0 1.0]

# computation of QR factorization

F = qrfact(A)

# Mathematical relation
F[:Q]*F[:R]== A
```

3. Target eigen-value , stopping criterion and locking ,purging

Definition

Deflation : Deflation is an important concept in the practical implementation of the QR iteration and therefore equally important to the IRAM. In the context of the QR iteration, deflation amounts to setting a small subdiagonal element of the Hessenberg matrix H to zero. This is called deflation because it splits the Hessenberg matrix into two smaller subproblems which may be independently refined further.

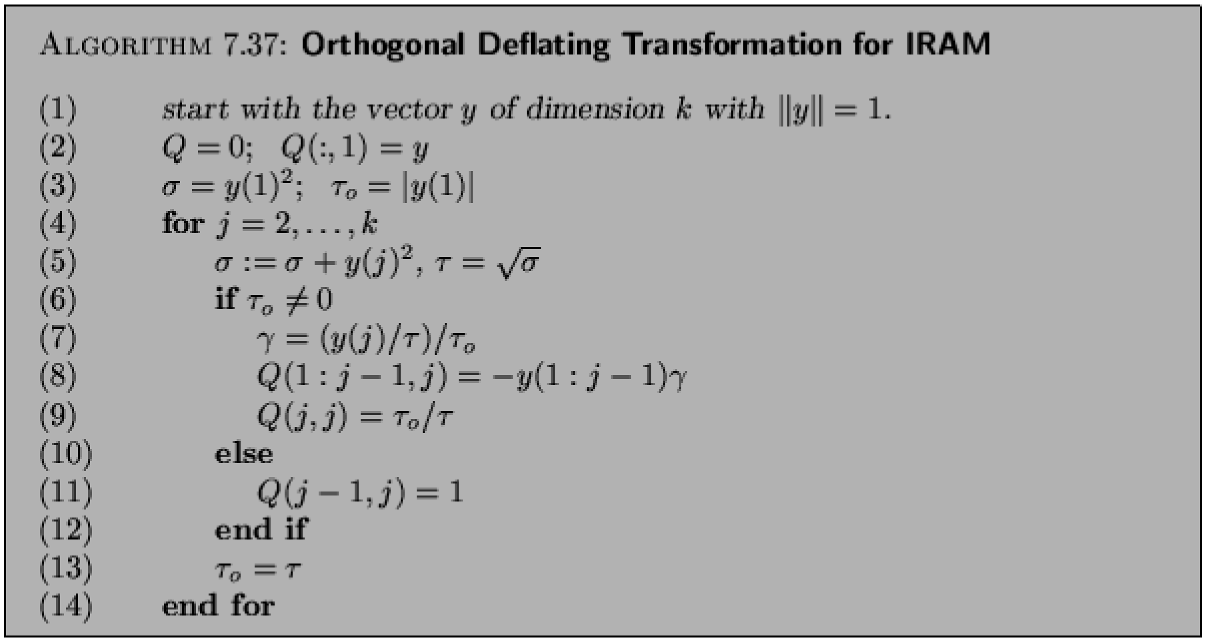
In the Arnoldi procedure, as with a QR iteration, it is possible for some of the leading k subdiagonals to become small during the course of implicit restarting. However, it is usually the case that there are converged Ritz values appearing in the spectrum of H long before small subdiagonal elements appear. This convergence is usually detected through observation of a small last component in an eigenvector y of H. When this happens, we are able to construct an orthogonal similarity transformation of H that will give an equivalent Arnoldi factorization with a slightly perturbed H that does indeed have a zero subdiagonal, and this is the basis of our deflation schemes.

In the context of IRAM, there are two types of deflation required:

Locking : If a Ritz value θ has converged (meaning $||Ax - x\theta|| < \epsilon D$) and thought to be a member of the wanted set of eigenvalues, then we wish to declare it converged, decouple the eigenpair (x, θ), and continue to compute remaining eigenvalues with no further alteration of x or θ. This process is called locking. more info (http://www.netlib.org/utk/people/JackDongarra/etemplates/node229.html)
Purging : If a Ritz value θ has converged but is not a member of the wanted set, then we wish to decouple and remove the eigenpair (x, θ) from the current Krylov subspace spanned by the Arnoldi vectors. This process is called purging. more info (http://www.netlib.org/utk/people/JackDongarra/etemplates/node230.html)

4.Orthogonal Deflating Transformation

A special orthogonal transformation to implement these deflation schemes. The deflation schemes are related to an eigenvector associated with a Ritz value that is to be deflated (either locked or purged). Given a vector y of unit length, Algorithm computes an orthogonal matrix Q such that $Qe_1 = y$ (hence $y = Q^* e_1$).The algorithm is shown below



ALGORITHM 7.37: **Orthogonal Deflating Transformation for IRAM**

```
(1)  start with the vector $y$ of dimension $k$ with $\|y\| = 1$.
(2)  $Q = 0;\ \ Q(:,1) = y$
(3)  $\sigma = y(1)^2;\ \ \tau_o = |y(1)|$
(4)  for $j = 2, \ldots, k$
(5)      $\sigma := \sigma + y(j)^2,\ \tau = \sqrt{\sigma}$
(6)      if $\tau_o \neq 0$
(7)          $\gamma = (y(j)/\tau)/\tau_o$
(8)          $Q(1:j-1, j) = -y(1:j-1)\gamma$
(9)          $Q(j,j) = \tau_o/\tau$
(10)     else
(11)         $Q(j-1, j) = 1$
(12)     end if
(13)     $\tau_o = \tau$
(14) end for
```

Further Details can be found at info (http://www.netlib.org/utk/people/JackDongarra/etemplates/node227.html#lckform)

5. Extending the Implementation for generalised eigen value problem `Ax = λBx`

Typically this takes the form

$$Ax = \lambda Bx$$

Basis functions that provide sparsity are usually not orthogonal in the natural inner product and hence, B is usually not diagonal. Thus it is typical for large scale eigenproblems to arise as generalized rather than standard problems with B symmetric and positive semi-definite. The matrix A is generally symmetric when the underlying continuous operator is self-adjoint and nonsymmetric otherwise. There are a number of ways to convert the generalized problem to standard form. There is always motivation to preserve symmetry when it is present.

We can achieve by If B is positive definite then factor $B = LL^H$ and the eigenvalues of $\hat{A} = L^{-1}AL^{-H}$ are the eigenvalues of (A, M) and the eigenvectors are obtained by solving $L^H x = \hat{x}$ where $\hat{x}$is an eigenvector of $\hat{A}$. This standard transformation is fine if one wants the eigenvalues of largest magnitude and it preserves symmetry if A is symmetric. However, when B is ill-conditioned this can be a dangerous transformation leading to numerical dificulties. Since a matrix factorization will have to be done anyway, one may as well formulate a **spectral transformation** if one were interested in computing the smallest eigenvalues of a σ

$$Ax = \lambda Bx$$
$$(A - \sigma B)x = Bx(\lambda - \sigma)$$
$$(A - \sigma B)^{-1}Bx = xv, \quad where\ v = \frac{1}{\lambda - \sigma}$$

ion. Eigenvalues λ that are near σ will be transformed to eigenvalues

$$v = \frac{1}{\lambda - \sigma}$$

that are at the extremes and typically well separated from the rest of the transformed spectrum.The corresponding eigenvectors remains unchanged.

Information link link (http://www.caam.rice.edu/software/ARPACK/UG/node33.html#shinv)

Timeline (tentative)

Community Bonding period (22nd April - 14th May)

My summer vacation will start on 30th of April. During this period, I would want to get myself more familiarized with the source code of **IterativeSolvers.jl** . I would devote my time trying to solve issues, I believe solving the above issue would help me strengthen my understanding of source code and make myself comfortable with contributing to the package.

Week 1

Goal: *Implement arnoldi factorization for* $Ax = \lambda x$

I have already implemented a basic version of the procedure mentioned above.Will work towards implementing it in an efficient and user-friendly (with proper documentation).Design of the function will mimic the interface for $eigs$ or Jacobi-Davidson packages.

Week 2 and 3

Goal: *Implement the IRA Algorithm*

During these weeks I will add IRA Algorithm, This involves two main parts figuring out the 'qrfact()' algorithm for sparse matrix and then implement a restart mechanism for convergence towards wanted eigenvalue.Above discussion on usage of wrapper around LAPACK.

Week 4 and 5

Goal: *Add support for different selection criteria for shifts and work on convergence criterion*

During this week, I would work to add functionality for the user to target different types of eigenvalues for example largest real part, smallest real part, largest imaginary part, smallest imaginary part, largest magnitude eigenvalues and interior eigenvalues using harmonic Ritz value.

Week 6 and 7

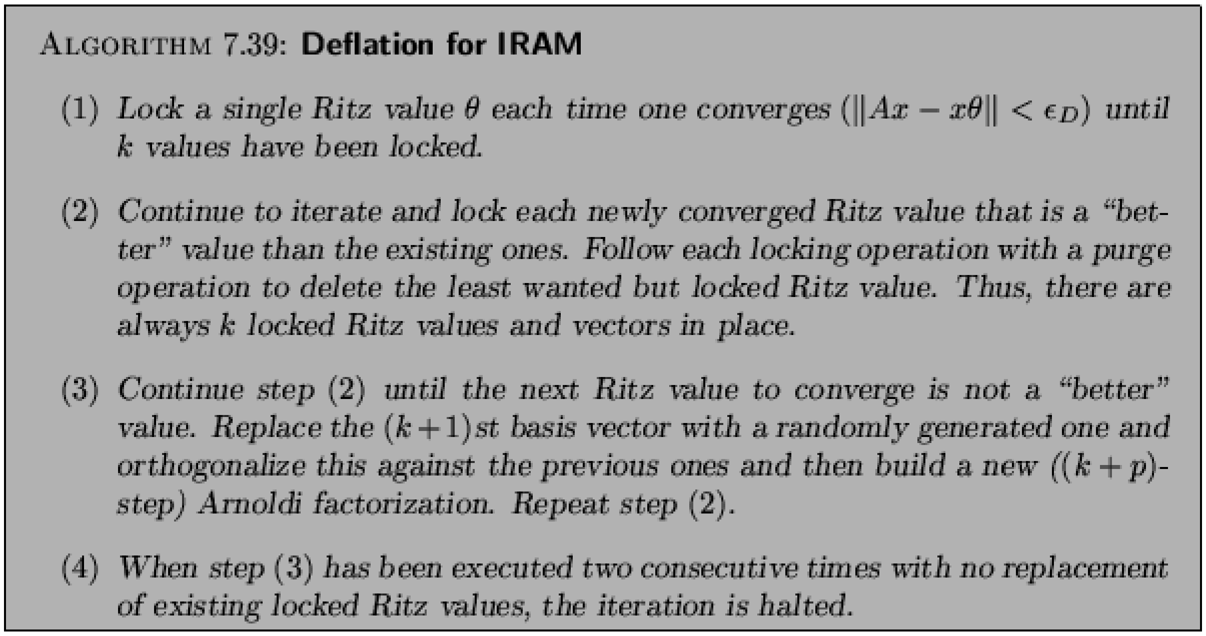
Goal: *Implement Orthogonal Deflating Transformation+Documentation*

During these 2 weeks, I implement a new function which would add Orthogonal Deflating Transformation functionality for IRAM.jl package, as well document and clean up my code base for the package.

Week 8

Goal: *Add Locking and Purging functionality for Ritz Pair*

During this week, I would be writing codes to use Locking and Purging for IRAM which has been verbally described below as :



ALGORITHM 7.39: **Deflation for IRAM**

(1) *Lock a single Ritz value θ each time one converges ($\|Ax - x\theta\| < \epsilon_D$) until k values have been locked.*

(2) *Continue to iterate and lock each newly converged Ritz value that is a "better" value than the existing ones. Follow each locking operation with a purge operation to delete the least wanted but locked Ritz value. Thus, there are always k locked Ritz values and vectors in place.*

(3) *Continue step (2) until the next Ritz value to converge is not a "better" value. Replace the $(k+1)$st basis vector with a randomly generated one and orthogonalize this against the previous ones and then build a new ($(k+p)$-step) Arnoldi factorization. Repeat step (2).*

(4) *When step (3) has been executed two consecutive times with no replacement of existing locked Ritz values, the iteration is halted.*

Week 9 and 10

Goal: *Solver For* $Ax = \lambda Bx$ *,Invert Spectral Transformation an eigenvector purification*

During these two weeks, I plan to implement Shift and Invert Spectral Transformation Mode to solve the general eigenvalue problem, a great C++ implementation for reference is Spectra (https://spectralib.org/) which mimics the ARPACK routines, I have already started with experimenting with code (https://github.com/yixuan/spectra), plan to implement on similar lines.

Week 11 and Beyond

Goal: *Eigenvector/Null-Space Purification, Benchmark and Documentation*

During this week , I plan to work on understanding and implementing Eigenvalue purification, as well as add documentation and benchmark results and compile code it into a repository.

At the end of this week, the Iterative Solver in IRAM.jl would be ready for testing for the Julia community.

End-Term evaluation

Goal: *Working towards the publication and final touches*

Buffer period of 1 week for any lagging work is kept. I also aim to work towards writing a blog of my work and getting my work published.

Note 1: I have no major plans for summer. I will try my best to learn as much as I can, during the project. My summer vacation starts on 30th April. Hence, I will start coding a 2 weeks earlier than the GSoC coding period, effectively giving me **14 weeks** to complete the project. Though my classes start in mid-July, it won't be an issue as I won't have any tests or examination until the end of August.

References

1. ARPACK (http://www.caam.rice.edu/software/ARPACK/UG/node45.html#SECTION00800000000000000000)
2. Slides (http://people.bath.ac.uk/mamamf/talks/talk220405.pdf)
3. Book (http://people.inf.ethz.ch/arbenz/ewp/Lnotes/lsevp.pdf)
4. Book-2 (http://www.netlib.org/utk/people/JackDongarra/etemplates/book.html)