Finding Rightmost Eigenvalues of Large Sparse Non-symmetric Parameterized Eigenvalue Problems

Minghao Wu
Applied Mathematics and Scientific Computation Program
Department of Mathematics
University of Maryland, College Park, MD
mwu@math.umd.edu

Advisor: Professor Howard Elman
Department of Computer Sciences
University of Maryland, College Park, MD
elman@cs.umd.edu

Abstract

This report has four main parts. In the first part, I state the eigenvalue problem I am trying to solve in my project, give a brief introduction to its background and application, and analyze the computational difficulties. In the second part, I explain the methods I use to solve the problem, mainly the eigenvalue solvers and matrix transformations. In the third part, I present the results I have obtained and the codes I develop to implement the methods in the previous part. In the last part of my report, I give a outline of what I will do in the future (AMSC 664).

Introduction

Consider the eigenvalue problem

$$A_S x = \lambda B_S x \tag{1}$$

where $\boldsymbol{A_S}$ and $\boldsymbol{B_S}$ are large sparse non-symmetric real $\boldsymbol{N \times N}$ matrices and $\boldsymbol{S}$ is a set of parameters given by the underlying Partial Differential Equation (PDE). For simplicity, I will drop the subscript $\boldsymbol{S}$ in the following discussion. People are interested in computing its rightmost eigenvalues (namely, eigenvalues with the largest real parts). The motivation lies in the determination of the stability of steady state solutions of non-linear systems of the form

$$B\frac{du}{dt} = f(u) \quad f : R^N \to R^N, u \in R^N \tag{2}$$

with large N and where u represents a state variable (velocity, pressure, temperature, etc). B is often called the mass matrix. Define the Jacabian matrix for the steady state u^* by $A=df/dx(u^*)$, then u^* is stable if all the eigenvalues of (1) have negative real parts. Typically, f arises from the spatial discretization of a PDE. Interesting applications of this kind occur in stability analyses in fluid mechanics, structural engineering and chemical reactions. The problem of finding rightmost eigenvalues also frequently occurs in Markov chain models, economic modeling, simulation of power systems and magnetohydrodynamics. When finite differences are used to discretize a PDE, then often $B=I$ and (1) is called a standard eigenproblem. If the equations are discretized by finite elements, then the mass matrix $B\neq I$ and (1) is called a generalized eigenvalue problem. For problems arising from fluid mechanics, B is often singular.

Major computational difficulties of this kind of problems are: (1) both A and B are large and sparse, so the algorithm we use must be efficient in dealing with large systems; (2) in many applications, the rightmost eigenvalues are complex, so we must consider complex arithmetic; (3) B is often singular, so it will give rise to spurious eigenvalues.

Beside the numerical algorithm in computing the rightmost eigenvalues of (1), how the parameter set S gives rise to the bifurcation phenomena, i.e, the steady state solution exchanges in stability, is also of people's interest. Examples are the Rayleigh number in nonlinear diffusion equation (Olmstead model) and the Damköhler number in the tubular reactor model. As the parameters vary, the rightmost eigenvalues might cross the imaginary axis, thus the steady state solution becomes unstable.

Methodology

Eigenvalue Solvers

Since both A and B are large and sparse, direct methods such as the QZ-algorithm for the generalized problem and the QR-algorithm for the standard problem are not feasible. A more efficient approach is the solution of the standard eigenvalue problem $Tx = \theta x$, which is a transformation of $Ax = \lambda Bx$, by iterative methods like Arnoldi's method, subspace iteration and Lanczos' method. Another reason why iterative methods are more suitable for this type of problem is that we are usually not interested in computing the full spectrum of the eigenvalue problem (1). Instead, we are only interested in computing a small set of the spectrum, which, in the project, is the rightmost eigenvalues. The eigensolver I want to explore in my project is Arnoldi's algorithm and its variants, such as the Implicitly Restarted Arnoldi algorithm.

1. Basic Arnoldi Algorithm

Arnoldi algorithm is an iterative eigensolver based on Krylov subspaces. Given a matrix ***A*** and a vector $\boldsymbol{u_1}$, a ***k***-dimensional Krylov subspace is spanned by the columns of the following matrix:

$$K_k(A,u_1)=\left[u_1\ Au_1\ A^2u_1\cdots A^{k-1}u_1\right]$$

provided that they are linear independent. ***k+1*** steps of Arnoldi algorithm give us the following decomposition:

$$AU_k = U_kH_k + \beta_k u_{k+1}e_k^T \tag{3}$$

where $\boldsymbol{U_k}$ is an orthonormal basis of the ***k***-dimensional Krylov subspace and [$\boldsymbol{U_k}$ $\boldsymbol{u_{k+1}}$] is an orthonormal basis of the ***k+1***-dimensional Krylov subspace, $\boldsymbol{H_k}$ is a ***k×k*** upper Hessenberg matrix, β_k is a scalar and e_k is a ***k×1*** vector $[0\,0\cdots 0\,1]$.

One thing worth mentioning is that we usually select ***k*** such that ***k«N***.
Premultiply (3) by $\boldsymbol{U_k^T}$:

$$U_k{}^T AU_k = U_k{}^T U_k H_k + \beta_k U_k{}^T u_{k+1}e_k^T = H_k\,. \tag{4}$$

Therefore, $\boldsymbol{H_k}$ is the projection of ***A*** onto the ***k***-dimensional Krylov subspace. We hope the eigenvalues of $\boldsymbol{H_k}$ can be good approximation of those of ***A*** because if that is true, we can solve a much smaller eigenvalue problem instead. Suppose ($\boldsymbol{\lambda,z}$) is an eigenpair of $\boldsymbol{H_k}$. Mutiply (4) by ***z***:

$$U_k{}^T A(U_k z) = H_k z = \lambda z = \lambda\left(U_k{}^T U_k\right)z = U_k{}^T \lambda(U_k z). \tag{5}$$

Then ($\boldsymbol{\lambda, U_k z}$) is an approximation of an eigenpair of ***A*** and we define the residual to be $\boldsymbol{A(U_k z) - \lambda(U_k z)}$.We can also obtain from (5) the following:

$$U_k{}^T\left(A(U_k z) - \lambda(U_k z)\right) = 0\,.$$

This tells us the residual is always orthogonal to the ***k***-dimensional Krylov subspace. Fig. 1 illustrate $\boldsymbol{A(U_k z)}$, $\boldsymbol{\lambda(U_k z)}$ and their residual:

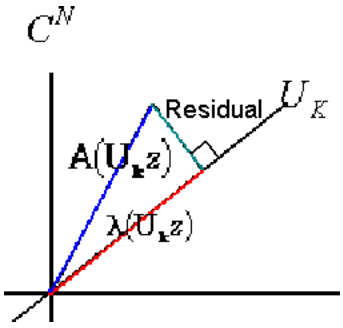

Fig. 1

As k increases, the residual will decrease and eventually, when $k=N$, $A(U_k z) = \lambda(U_k z)$, which means $(\lambda, U_k z)$ is an exact eigenpair of A.
Though Arnoldi algorithm is powerful, it is not a good idea to apply it naïvely to our eigenvalue problem. There are basically two reasons: (1) since we are dealing with large systems, increasing k to improve the performance of Arnoldi is not practical. For example, when A is of size 10,000×10,000 and k = 100, then we need 10 megabytes to store the Krylov basis U_{100} to double precision; (2) in many real world applications, matrix B is singular. This will give rise to spurious eigenvalues if we don't do anything clever. A lot of variants of Arnoldi algorithm follow the line of restarting the Arnoldi process by using a more carefully chosen starting vector u_1 (in the basic Arnoldi algorithm, u_1 is randomly chosen). In my project I will explore one of them, the Implicitly Restarted Arnoldi algorithm (IRA).

2. Implicitly Restarted Arnoldi (IRA) Algorithm

The basic idea of IRA is to filter out unwanted eigendirections from the original starting vector u_1 by using the most recent spectrum information and also a clever filtering technique.

Suppose we first use Arnoldi algorithm and find m ($m>k$) approximated eigenpairs:

$$(\mu_1, x_1),\ (\mu_2, x_2),\ \dots,\ (\mu_m, x_m)$$

($\mathrm{Re}(\mu_i) \geq \mathrm{Re}(\mu_j)$, $i<j$) and the following Arnoldi decomposition:

$$AU_m = U_m H_m + \beta_m u_{m+1} e_m^T . \tag{6}$$

We want to filter out the eigendirection $x_{k+1}, \dots, x_m$ from the starting vector u_1 so that it can be richer in the k eigendirections we are interested in. Suppose now we want to filter out eigendirection x_{k+1} from the starting vector.
We first subtract $\mu_{k+1} U_m$ from both sides of (6):

$$(A - \mu_{k+1} I) U_m = U_m (H_m - \mu_{k+1} I) + \beta_m u_{m+1} e_m^T$$

then compute the QR decomposition of $H_m - \mu_{k+1} I$ (suppose $H_m - \mu_{k+1} I = Q_1 R_1$, Q_1 is orthonormal and R_1 is upper triangular):

$$(A - \mu_{k+1} I) U_m = U_m Q_1 R_1 + \beta_m u_{m+1} e_m^T , \tag{7}$$

next, multiply both sides of (7) by Q_1:

$$(A - \mu_{k+1} I)(U_m Q_1) = (U_m Q_1)(R_1 Q_1) + \beta_m u_{m+1} e_m^T Q_1 , \tag{8}$$

and add $\mu_{k+1} U_m Q_1$ to both sides of (8):

$$A(U_m Q_1) = (U_m Q_1)(R_1 Q_1 + \mu_{k+1} I) + \beta_m u_{m+1} e_m^T Q_1 .$$

This way we have a new Arnoldi decomposition:

$$AU_m^{(1)} = U_m^{(1)} H_m^{(1)} + \beta_m u_{m+1} e_m^T Q_1$$

where $U_m^{(1)} = U_m Q_1$ and $H_m^{(1)} = R_1 Q_1 + \mu_{k+1} I$ results from one **QR** step with shift μ_{k+1} applied to H_m. The first column of $U_m^{(1)}$ is proportional to $(A - \mu_{k+1} I)u_1$, so the unwanted eigendirection x_{k+1} has been filtered out from the starting vector. We repeat the process with $\mu_{k+2},\ldots,\mu_m$ and end up with the new Krylov basis $U_m^{(m-k)}$ whose first column is proportional to $(A - \mu_{k+1} I)\ldots(A - \mu_m I)u_1$. All the unwanted eigendirections have been filtered out from the original starting vector u_1. So we don't need to restart Arnoldi process explicitly, instead, we apply the shifted **QR** algorithm to the upper Hessenberg matrix H_m to obtain a new Arnoldi decomposition which is equivalent to the decomposition we will obtain if we start the Arnoldi process with the filtered vector. That's why this method is called Implicitly Restarted Arnoldi. A typical IRA circle consists of the following three steps:

1. Compute m eigenpairs of H_m ($k < m \ll N$)
2. Apply Shifted **QR** algorithm $m-k$ times (with shifts $\mu_{k+1},\ldots,\mu_m$) to H_m to compute a new Arnoldi decomposition with filtered starting vector
3. Go back to 1 if the k wanted eigenpairs do not converge

Matrix Transformation

Matrix transformation is crucial in solving problems like (1). There are two important reasons for this approach. First, a practical reason is that iterative methods like Arnoldi's method and subspace iteration cannot solve generalized eigenvalue problems, which makes a transformation necessary. A second reason is of a numerical nature. It is well known that iterative eigenvalue solvers applied to A quickly converge to the well-separated extreme eigenvalues of A. When A arises from the spatial discretization of a PDE, then the rightmost eigenvalues of A are in general not well separated. This implies slow convergence. The iterative method may converge to a wrong eigenvalue. Instead, one applies eigenvalue solvers to a transformation T with the aim of transforming the rightmost eigenvalues of A to well-separated extremal eigenvalues of T, which are easily found by the eigenvalue solvers we consider. I will explore two kinds of matix transformation: Shift-invert transformation and Cayley transformation.

1. Shift – Invert Transformation

The definition of shift – invert transformation is as following:

$$T_{SI}(A, B; \sigma) = (A - \sigma B)^{-1} B$$

where σ is called the shift. After the transformation, we solve the standard eigenvalue problem

$$T_{SI} x = \theta x$$

instead of the generalized eigenvalue problem $Ax = \lambda Bx$. After that, we use the relationship between λ and θ:

$$\lambda \quad \rightarrow \quad \theta = \frac{1}{\lambda - \sigma}$$

to recover the eigenvalue of the original problem.

Shift – invert transformation maps $\boldsymbol{\lambda}$ that are close to $\boldsymbol{\sigma}$ away from origin and maps $\boldsymbol{\lambda}$ far from $\boldsymbol{\sigma}$ close to origin. Fig. 2 and Fig. 3 illustrate this property.

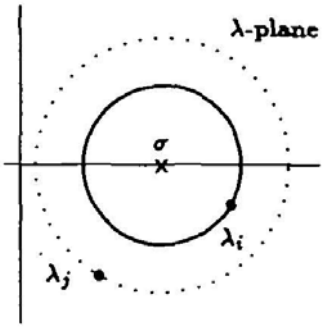

Fig. 2

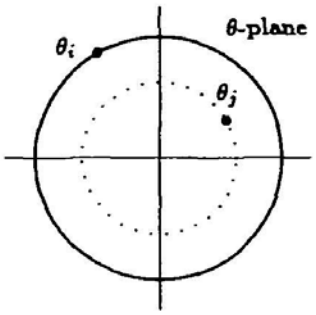

Fig. 3

So an obvious choice of $\boldsymbol{\sigma}$ is an approximation of the eigenvalue we want to compute.

2. Cayley Transformation

The Cayley transformation is defined by

$$T_C(A, B; \sigma, \tau) = I + (\sigma - \tau)T_{SI} = (A - \sigma B)^{-1}(A - \tau B)$$

where $\boldsymbol{\sigma}$ is called the shift and $\boldsymbol{\tau}$ is called the anti-shift. After the transformation, we solve the standard eigenvalue problem

$$T_C x = \theta x$$

instead of the generalized eigenvalue problem $\boldsymbol{Ax = \lambda Bx}$. After that, we use the relationship between $\boldsymbol{\lambda}$ and $\boldsymbol{\theta}$:

$$\lambda \quad \rightarrow \quad \theta = \frac{\lambda - \tau}{\lambda - \sigma}$$

to recover the eigenvalue of the original problem.

Cayley transformation maps $\boldsymbol{\lambda}$ close to $\boldsymbol{\sigma}$ to $\boldsymbol{\theta}$ away from the unit circle and maps $\boldsymbol{\lambda}$ close to $\boldsymbol{\tau}$ to $\boldsymbol{\theta}$ with small modulus. That's why $\boldsymbol{\tau}$ is called the anti-shift. The most interesting property is that the line $\boldsymbol{Re(\lambda) = (\sigma+\tau)/2}$ is mapped to the unit circle, and $\boldsymbol{\lambda}$ to the left of the line are mapped inside the unit circle, $\boldsymbol{\lambda}$ to the right of the line are mapped outside the unit circle. Fig.4 and Fig. 5 demonstrate this property.

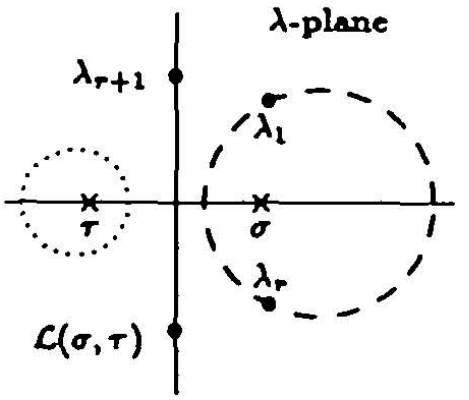

Fig. 4

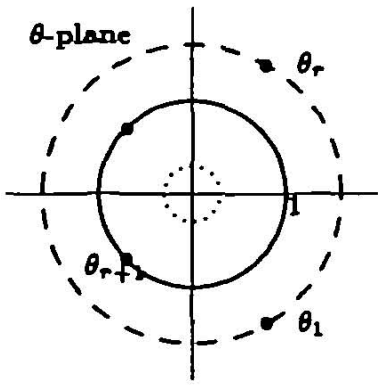

Fig. 5

So if we are interested in computing r rightmost eigenvalues, we should choose σ and τ such that λ_{r+1} lies on the line $Re(\lambda) = (\sigma+\tau)/2$.

Discretization of PDEs

In this project, finite difference and finite element method will be used to discretize the PDEs. They are the most commonly used methods in the discretization of PDEs.

Complex Arithmetic

Since the rightmost eigenvalues are complex in many real-world problems, complex arithmetic is considered in this project.

Mid-Year Progress Report

Overview

Time	Schedule	Progress
AMSC 663		
Before November	solve the first test problem	√
	explore the effect of Rayleigh number in the problem	√
November	solve the second test problem	-
	explore the effect of Damköhler number in the problem	-
December	modify the Implicitly Restarted Arnoldi Algorithm	√
	solve the third test problem	√

	finish midterm report	√
	give midterm presentation	√
AMSC 664		
January & February	implement iterative linear system solver	×
March	give mid-term presentation	×
	discretize a Navier-Stokes equation	×
	Solve the eigenvalue problem arises from the discritization	×
April	explore the effect of the parameter in the last problem	×
	Start writing final report	×
May	Finish final report	×
	Give final presentation	×

√: finished -: in progess ×: not started

The First Test Problem: Olmstead Model

1. Problem Statement

The first test problem is from a paper by Olmstead *et al* in 1986. The model is governed by a coupled system of PDEs:

$$\begin{cases} u_t = S_{xx} + cu_{xx} + Ru - u^3 \\ bS_t = (1-c)u - S \end{cases}$$

with boundary condition:

$$u = S = 0,\ x = 0,\ \pi.$$

This model represents the flow of a layer of viscoelastic fluid heated from below. ***u*** is the speed of the fluid, ***S*** is a quantity related to viscoelatic forces, and ***b***, ***c***, ***R*** are all scalars. In particular, ***R*** is called the Rayleigh number and is the parameter I study in this problem.

2. Discretization of PDEs

I use second order centered difference method to discretize the system of PDEs. Grid size is ***h = 1/(N/2)***. Order the unknowns by grid points, i.e:

$$y^T = [u_1\ \ S_1\ \ u_2\ \ S_2 \cdots u_{N/2}\ \ S_{N/2}].$$

Then the system can be written as ***dy/dt = f(y)***. Matrix B in this case is identity. We evaluate the Jacobian matrix ***A*** at the trivial (all 0) steady state solution ***y*: A = dy/dt(y*)*** with ***N*** = 1000, ***b*** = 2, ***c*** = 0.1 and ***R*** = 0.6. The resulting matrix is a large (500×500) sparse nonsymmetric matrix with bandwidth 6. Fig. 6 illustrate its structure:

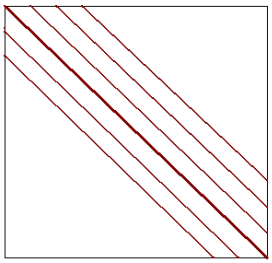

Fig. 6

The nonzero elements of the matrix all sit on the six bands in the middle. This kind of matrix structure is good for linear system solvers. That's why I order the unknowns by the grid points.

3. Matlab Implementation of Arnoldi Algorithm

To implement Arnoldi algorithm together with the shift-invert matrix transformation, I write a Matlab routine:

[v,X,U,H]=SI_Arnoldi(A,B,k,sigma).

The input of the routine is:

A, ***B***: matrix ***A*** and ***B*** in the problem $\boldsymbol{Ax = \lambda Bx}$;

k: the number of eigenpairs wanted;

sigma: the shift $\boldsymbol{\sigma}$ in shift-invert transformation.

The output of the routine is:

v: a vector of ***k*** computed eigenvalues;

X: a matrix whose columns are the ***k*** eigenvectors;

U: the orthonormal basis of the ***k+1***-dimensional Krylov subspace;

H: the ***k×k*** upper Hessenberg matrix.

4. Computational Result

Use ***k = 10*** and ***sigma*** = -0.549994+2.01185i. As discussed in the "Matrix Transformation" section, a good choice of σ is an approximation of the eigenvalue of interest. When ***b*** = 2, ***c*** = 0.1 and ***R*** = 0.3, the rightmost eigenvalue is -0.549994±2.01185i. So -0.549994+2.01185i can be viewed as an approximation to the rightmost eigenvalue when ***R*** = 0.6.

The computational result is:

- rightmost eigenvalues: $\boldsymbol{\lambda_{1,2}} = 0 \pm 0.4472i$
- residual: $\|\boldsymbol{A\lambda_i - \lambda_i x_i}\| = 8.4504e\text{-}012$, $\boldsymbol{i} = 1,2$

and the result agrees with the literature.

I also compute the critical Rayleigh number. Critical number Rayleigh number is the value of ***R*** under which the rightmost eigenvalue just cross the imaginary axis

and thus makes the steady state solution change from stable to unstable. To compute the critical Rayleigh number $\boldsymbol{R_C}$, I fix $\boldsymbol{b}$ and $\boldsymbol{c}$, start from a large $\boldsymbol{R}$, compute the rightmost eigenvalue, and then decrease $\boldsymbol{R}$, compute the rightmost eigenvalue again using the rightmost eigenvalue of the previous step as the $\boldsymbol{\sigma}$, repeat this process until the rightmost eigenvalue become negative. Fig. 7 shows the value of $\boldsymbol{R_C}$ under different values of $\boldsymbol{b}$ and $\boldsymbol{c}$.

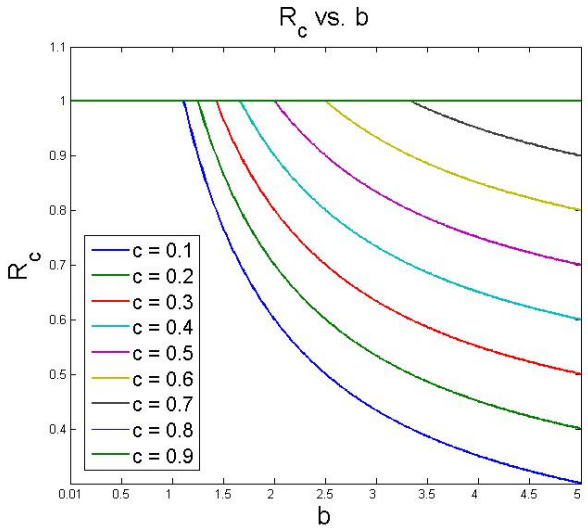

Fig. 7

The graph shows that $\boldsymbol{R_C}$ follows the rule:

$$R_C = \begin{cases} 1 & b \le \dfrac{1}{1-c} \\ \dfrac{1}{b} + c & b > \dfrac{1}{1-c} \end{cases}$$

which also agrees with the literature.

The Third Test Problem

1. Problem Statement

This test problem is from a Paper by Meergergen and Spence (1995). Consider the following eigenvalue problem:

$$\begin{bmatrix} K & C \\ C^T & 0 \end{bmatrix} x = \lambda \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} x$$

where $\boldsymbol{K}$ is a 200×200 matrix, $\boldsymbol{C}$ is a 200×100 matrix, and $\boldsymbol{M}$ is a 200×200 matrix. They are all of full rank. $\boldsymbol{K}$, $\boldsymbol{C}$ and $\boldsymbol{M}$ are generated by using Matlab function "rand".

Although this problem is generated artificially, eigenvalue problem with this kind of block structure appears in the stability analysis of steady state solution of Navier – Stokes equations for incompressible flow. Eventually, I want to solve the eigenvalue problem arises from the discretization of Navier – Stokes equations.

2. Matlab Implementation

I use two algorithms to solve this problem and compare their performance. The first one is the basic Arnoldi algorithm and the second one is the Implicitly Restarted Arnoldi (IRA) algorithm. To implement the first algorithm, I again use the Matlab code in the first problem. For the IRA, I use a code written by Fei Xue. The IRA code has the following important functions:

Name of the function	**Description**
IRADirectMain	Main function
ArnoldiExpd	Given a k – step Arnoldi decomposition,expands it to an m – step Arnoldi decomposition ($m \geq k$)
sglshiftQR, dblshiftQR	Implements the shifted QR algorithm (sglshiftQR if the shift is real, dblshiftQR if the shift is complex)
contractionIRA	Given an m – step Arnoldi decomposition, contracts it to a k –step Arnoldi decomposition($k \leq m$)
extractEigenPairs	Outputs the eigenpairs of interest and also the error

Fig. 8 illustrates the basic structure of the IRA code.

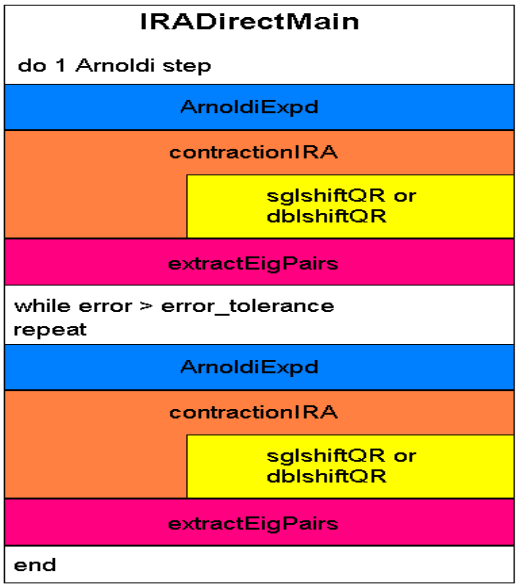

Fig. 8

3. Computational Result

I first use Matlab function "eig" to compute the exact eigenvalues of the system, and then use Arnoldi algorithm and IRA to compute 10 rightmost eigenvalues. The shift σ for both Arnoldi and IRA is 60. Fig. 9 shows their results:

Exact Eigenavalues	Computed Eigenavalues (IRA)	Computed Eigenavalues (Arnoldi)
49.9129 + 0i	49.9129 + 0i	49.9129 + 0i
2.9112 + 1.1256i	2.9112 - 1.1256i	193.8412 - 7113830.9524i
2.9112 - 1.1256i	2.9112 + 1.1256i	193.8412 + 7113830.9524i
2.5036 + 0.0624i	2.5036 - 0.0624i	3.0891 + 0i
2.5036 - 0.0624i	2.5036 + 0.0624i	2.6112 + 0i
2.3792 + 0i	2.3792 + 0i	-0.5752 - 2.7079i
2.1318 + 0.9356i	2.1318 - 0.9356i	-0.5752 + 2.7079i
2.1318 - 0.9356i	2.1318 + 0.9356i	-1.0901 - 0.7532i
2.1081 + 1.3539i	2.1081 - 1.3539i	-1.0901 + 0.7532i
2.1081 - 1.3539i	2.1081 + 1.3539i	-47.3106 + 0i

Fig. 9

The IRA gives very accurate result. But Arnoldi gives rise to spurious eigenvalues, which are highlighted by red color. The computed eigenvalues highlighted by blue color are approximation of the true eigenvalues, but unfortunately, they are not the 10 rightmost eigenvalues. Not only does the Arnoldi give rise to spurious eigenvalues, it also computes the wrong eigenvalues. It is obvious that IRA outperforms Arnoldi.

Future Work (AMSC 664)

1. Finish test problem 2: tubular reactor model
2. Implement iterative linear system solvers and compare with direct solver
3. Implement Cayley matrix transformation and compare with Shift - invert
4. Apply the algorithm to Navier – Stokes equations

Refernce

1. Meerbergen, K & Roose, D 1996 Matrix transformation for computing

rightmost eigenvalues of large sparse non-symmetric eigenvalue problems. *SIAM J. Numer. Anal.* 16, 297-346

2. Meergergen, K & Spence, A 1997 Implicitly restarted Arnoldi with purification for the shift-invert transformation. *Math. Comput.* 66, 667-698.
3. Olmstead, W. E., Davis, W. E., Rosenblat, S. H., & Kath, W. L. 1986 Bifurcation with memory. *SIAM J. Appl. Math.* 40, 171-188.
4. Stewart, G. W. 2001 Matrix algorithms, volume II: eigensystems. *SIAM*
5. Sorensen, D. C. 1992 Implicitly application of polynomial filters in a k – step Arnoldi Method. *SIAM J. Matrix Annal. Appl.* 13, 357-385
6. Meerbergen, K & Roose, D 1997 The restarted Arnoldi method applied to iterative linear system solvers for the computation of rightmost eigenvalues. *SIAM J. Matrix Anal. Appl.* 18, 1-20