University of Colorado Boulder

APPM 2360 - Intro to Differential Equations and Linear Algebra

Project 2

Eigenvalues in Plankton Blooms

Author:
Abdulla AlAmeri
ID: 109364560
Section 231

Author:
Robert Redfern
ID: 107219091
Section 549

Author:
Matthew Zola
ID: 107195871
Section 150

April 4, 2019

College of Engineering & Applied Science
UNIVERSITY OF COLORADO BOULDER

I. Introduction

Plankton populations can be modeled as a differential equation that takes into account both growth and diffusion (loss). The DE developed from the Kierstead and Slobodkin paper will be analyzed in this report to solve for the solutions to the DE. This report will also explore the QR Algorithm and the Power Method as a means of analyzing the eigenvalues of a large matrix.

II. The Plankton Growth Equation

The growth-diffusion equation for plankton is shown below with K being the specific growth rate over space and time. K is then multiplied by the concentration function c in order to represent a linear growth term. Concentration will equal zero at the end of the interval as well. The derivation below shows that the simplified form of the diffusion equation is acquired by substituting the definition of c(x,t) into the original growth model.

$$\frac{\partial c}{\partial t} = D\frac{\partial^2 c}{\partial x^2} + Kc \tag{1}$$

$$c(x,t) = u(x,t)e^{Kt} \tag{2}$$

$$\frac{\partial c}{\partial t} = \frac{\partial u}{\partial t}e^{Kt} + Ku(x,t)e^{Kt}$$

$$\frac{\partial^2 c}{\partial x^2} = e^{Kt}\frac{\partial^2 u}{\partial x^2}$$

$$\frac{\partial u}{\partial t}\cancel{e^{Kt}} + Ku(x,t)\cancel{e^{Kt}} = D\cancel{e^{Kt}}\frac{\partial^2 u}{\partial x^2} + Ku(x,t)\cancel{e^{Kt}}$$

$$\frac{\partial u}{\partial t} + \cancel{Ku(x,t)} = D\frac{\partial^2 u}{\partial x^2} + \cancel{Ku(x,t)}$$

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} \tag{3}$$

Fig. 1 Derivation of Equation (3) using Equations (1) and (2)

In order to solve the simplified diffusion equation while having D remain constant, u(t) is broken down as a product of T(t) and X(x). The eigenvalues of these functions, λ, are therefore the solutions of the diffusion equation. Below is the solution for both T(t) and X(x) using separation of variables.

$$
\begin{aligned}
T' + \lambda D T &= 0 \\
\frac{\mathrm{d}T}{\mathrm{d}t} &= -(\lambda D)T \\
\int \frac{1}{t}\mathrm{d}T &= \int -(\lambda D)\mathrm{d}t \\
T(t) = Ce^{-\lambda D t},\ T(0) &= 0 \\
\mathbf{T(t)} &= \mathbf{e^{-\lambda D t}} \\
T(x) = Ce^{-\lambda D x}, T(0) &= 0 \\
\alpha = \frac{-b}{2a} &= \frac{0}{2} = 0 \\
\beta = \frac{\sqrt{4\lambda}}{2} &= \sqrt{\lambda} \\
X(x) = \cancel{e^{0}}[c_1 cos(x\sqrt{\lambda}) + c_2 sin(x\sqrt{\lambda})] & \\
X(x) = c_1 cos(x\sqrt{\lambda}) + c_2 sin(x\sqrt{\lambda}) & \\
X(0) = 0 = c_1 cos(0) + c_2 sin(0) \rightarrow c_1 &= 0 \\
X(l) = 0 = c_2 sin(l\sqrt{\lambda}) \rightarrow c_1 &= 1 \\
\mathbf{X(x)} = \mathbf{sin(x\sqrt{\lambda})} &
\end{aligned}
$$

Fig. 2 Solving the separated ODE for $X(x)$ and $T(t)$

Thus, for each eigenvalue, λ_n, the eigenfunctions are $X_n(x) = sin(\sqrt{\lambda_n}x)$ and $T_n(t) = e^{-D\lambda_n t}$.

Equation (2) can be rewritten as a linear operator in the form $L = D\frac{\partial^2 u}{\partial x^2} - \frac{\partial u}{\partial t}$. To prove L is a linear operator, it must be closed under scalar addition and subtraction:

$$
\begin{aligned}
L(k_1u_1 + k_2u_2) &= 0 \\
D\frac{\partial^2 k_1u_1 + k_2u_2}{\partial x^2} - \frac{\partial k_1u_1 + k_2u_2}{\partial t} &= 0 \\
D\frac{\partial^2 k_1u_1}{\partial x^2} + D\frac{\partial^2 k_2u_2}{\partial x^2} - \frac{\partial k_1u_1}{\partial t} - \frac{\partial k_2u_2}{\partial t} &= 0 \\
k_1(D\frac{\partial^2 u_1}{\partial x^2} - \frac{\partial u_1}{\partial t}) + k_2(D\frac{\partial^2 u_2}{\partial x^2} - \frac{\partial u_2}{\partial t}) &= 0 \\
k_1\cancel{L(u_1)}^{0} + [k_2\cancel{L(u_2)}^{0} &= 0
\end{aligned}
$$

Fig. 3 Proof that Equation (2) can be written as a linear operator

Thus, the operator, L is linear and the sum of all of the X_nT_n must also be solutions due to the principle of superposition.

By plugging the solution for $u(x,t)$ into equation (2), the result is equation (6):

$$c(x,t) = u(x,t)e^{Kt}$$

$$u(x,t) = \sum_{n=1}^{\infty} A_n sin(\sqrt{\lambda_n}x)e^{-D\lambda_n t}$$

$$c(x,t) = \sum_{n=1}^{\infty} A_n sin(\sqrt{\lambda_n}x)e^{-D\lambda_n t}e^{Kt} = \sum_{n=1}^{\infty} A_n sin(\sqrt{\lambda_n}x)e^{(K-D\lambda_n)t} \quad (6)$$

Fig. 4 Derivation for Equation (6) for the concentration

If K > λ_1 then c(x,t) will contain a positive exponent in the e term. This means the concentration will grow. This will keep the derivative of the concentration function positive as well. The derivative of the concentration function shows if the population of plankton is increasing or decreasing over time as long as the space remains constant.

III. Discretization

The eigenvalues problem can be solved using various approaches. One approach is discretizing the problem in order to solve it. If the derivative operator was considered, an estimation for the derivative can be obtained using the first derivative approximation. Figure 5 shows how the forward and backward first derivatives estimations can be used to obtain an expression for the second derivative.

$$X'(x) = \frac{X(x) - X(x-h)}{h} \quad (3)$$

$$X'(x-h) = \frac{X(x) - X(x-h)}{h}$$

$$X''(x) = \frac{X'(x) - X'(x-h)}{h} = \frac{\frac{X(x)-X(x-h)}{h} - \frac{X(x)-X(x-h)}{h}}{h}$$

$$\mathbf{X''(x) = \frac{X(x-h) - 2X(x) + X(x+h)}{h^2}}$$

Fig. 5 Derivation for the second derivative using first derivative approximation

$$x_1 = X(h) = X''(h) = \frac{X(2h) - 2X(h) + X(0)}{h^2}$$

$$x_2 = X(2h) = X''(2h) = \frac{X(3h) - 2X(2h) + X(h)}{h^2}$$

$$x_3 = X(3h) = X''(3h) = \frac{X(4h) - 2X(3h) + X(2h)}{h^2}$$

Fig. 6 Equations for x1, x2, and x3 leading to the matrix equation

Once an expression for the second derivative is obtained, it can be treated as a matrix. By setting x = hk, We can consider the coefficients of the terms X(x-h), X(x), X(x+h) in a matrix. We recall that the step size h is taken to be constant, and hence it can be pulled outside the matrix. As a result, the matrix has that represents the following equation

$$-X" = \lambda X$$

Fig. 7 second derivative and eigenvalues in matrix

can be reduced to the following matrix:

$$\frac{1}{h^2}\begin{bmatrix} 2 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \ddots & 2 & -1 \\ & & & & & -1 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix}$$

Fig. 8 The expansion of the equation in figure 6

It can be seen that when x = 1, the coefficient of $-X''$ reduces to 0 2 -1 respectively for X(x-h), X(x), and X(x+h), for x = hk, and so on. A MATLAB code to create the generic matrix An is shown in appendix A. The step size h is $(n + 1/l)^2$ for l = 1, and hence the step size depends on the size of the matrix, n.

IV. QR Algorithm

The Gram-Schmidt algorithm is a process applied for a set of basis to create orthonormal basis out of the same set. In other words, Gram-Schmidt takes a set of bases and make all of them perpendicular to each other and normalize all of them. The QR factorization is factorization done on matrices to rewrite a matrix A as a multiplication of two to matrices, Q and R, and hence the name QR. The QR method is a very effective method to solve systems of equations. It is very powerful when the size of the matrix increases because it takes less time to do the computations. The QR method depends on the Gram-Schmidt process, because the matrix Q is constructed by applying the Gram-Schmidt process on the columns of a matrix under investigation, and then the matrix R can be obtained. In the special case where Q is a square matrix, it happens to be that Q^T Q = I (i.e. $Q^T = Q$-1), $and hence R simply is Q^T$ A.

We can prove the multiplication QR and RQ gives similar matrices. In other words, QR and RQ have the same eigenvalues. For instance, if we consider $A_{n=3}$ matrix and $A_{n=4}$ we can prove that they have the same eigenvalues. Considering the matrix $A_{n=3}$, it yields the following Q matrix:

$$\begin{pmatrix} \frac{2\sqrt{5}}{5} & \frac{3\sqrt{70}}{70} & \frac{\sqrt{14}}{14} \\ -\frac{\sqrt{5}}{5} & \frac{3\sqrt{2}\sqrt{35}}{35} & \frac{\sqrt{2}\sqrt{7}}{7} \\ 0 & -\frac{\sqrt{5}\sqrt{14}}{14} & \frac{3\sqrt{14}}{14} \end{pmatrix}$$

and the following R matrix

$$\begin{pmatrix} 16\sqrt{5} & -\frac{64\sqrt{5}}{5} & \frac{16\sqrt{5}}{5} \\ 0 & \frac{16\sqrt{5}\sqrt{14}}{5} & -\frac{128\sqrt{2}\sqrt{35}}{35} \\ 0 & 0 & \frac{32\sqrt{2}\sqrt{7}}{7} \end{pmatrix}$$

The multiplication QR results in the matrix:

$$\begin{pmatrix} 32 & -16 & 0 \\ -16 & 32 & -16 \\ 0 & -16 & 32 \end{pmatrix}$$

and the multiplication RQ results in the matrix

$$\begin{pmatrix} \frac{224}{5} & -\frac{16\sqrt{14}}{5} & 0 \\ -\frac{16\sqrt{14}}{5} & \frac{1312}{35} & -\frac{32\sqrt{5}}{7} \\ 0 & -\frac{32\sqrt{5}}{7} & \frac{96}{7} \end{pmatrix}$$

However, both of them give the same eigenvalues (rounded)

$$\begin{pmatrix} 54.63 & 32 & 9.37 \end{pmatrix}$$

Similarly, for the case of $A_{n=4}$, the matrix Q is:

$$, \begin{pmatrix} \frac{2\sqrt{5}}{5} & \frac{3\sqrt{70}}{70} & \frac{2\sqrt{105}}{105} & \frac{\sqrt{30}}{30} \\ -\frac{\sqrt{5}}{5} & \frac{3\sqrt{2}\sqrt{35}}{35} & \frac{4\sqrt{105}}{105} & \frac{\sqrt{2}\sqrt{15}}{15} \\ 0 & -\frac{\sqrt{5}\sqrt{14}}{14} & \frac{2\sqrt{3}\sqrt{35}}{35} & \frac{\sqrt{3}\sqrt{10}}{10} \\ 0 & 0 & -\frac{\sqrt{7}\sqrt{15}}{15} & \frac{2\sqrt{2}\sqrt{15}}{15} \end{pmatrix}'$$

and the matrix R is:

$$, \begin{pmatrix} 25\sqrt{5} & -20\sqrt{5} & 5\sqrt{5} & 0 \\ 0 & 5\sqrt{70} & -\frac{40\sqrt{7}\sqrt{10}}{7} & \frac{25\sqrt{5}\sqrt{14}}{14} \\ 0 & 0 & \frac{25\sqrt{7}\sqrt{15}}{7} & -\frac{100\sqrt{5}\sqrt{21}}{21} \\ 0 & 0 & 0 & \frac{25\sqrt{5}\sqrt{6}}{6} \end{pmatrix}'$$

the multiplication QR result is:

$$, \begin{pmatrix} 50 & -25 & 0 & 0 \\ -25 & 50 & -25 & 0 \\ 0 & -25 & 50 & -25 \\ 0 & 0 & -25 & 50 \end{pmatrix}'$$

and the multiplication RQ is:

$$, \begin{pmatrix} 70 & -5\sqrt{14} & 0 & 0 \\ -5\sqrt{14} & \frac{410}{7} & -\frac{125\sqrt{2}\sqrt{3}}{14} & 0 \\ 0 & -\frac{125\sqrt{2}\sqrt{3}}{14} & \frac{1150}{21} & -\frac{25\sqrt{2}\sqrt{7}}{6} \\ 0 & 0 & -\frac{25\sqrt{2}\sqrt{7}}{6} & \frac{50}{3} \end{pmatrix}'$$

yet both matrices have the same eigenvalues which are:

$$\begin{pmatrix} 90.4508 & 65.4508 & 34.5492 & 9.5492 \end{pmatrix}$$

Hence, it can be seen that the QR algorithms can be used as an iterative process to get a better approximation for eigenvalues. For the purpose of this investigation, we can run the algorithm up to a certain condition rather than certain tolerance. The QR algorithms were used 1000 times for matrices on An where n = 10, 20, 100. The algorithm (shown in Appendix) obtained the QR factorization for a matrix An, reversed the order of multiplication (i.e. RQ), and then repeated the same thing. The effectiveness of applying this iterative process on bigger matrices can be seen when compared with actual eigenvalue. An actual eigenvalue was given to be $\lambda_1 = (\pi/l)^2$, and for l = 1 , this simply reduces to $\lambda_1 = (\pi)^2$. When compared to λ_1, as the size of the matrix increased, the error went down dramatically. Table 1 and shows the obtained results for n = 10, 20, 100, which is represented graphically in figure 10

Table 1 The error compared to actual eigenvalue for different sized matrices

An	n = 10	n = 20	n = 100
Error	0.0669	0.0184	7.9572e-04

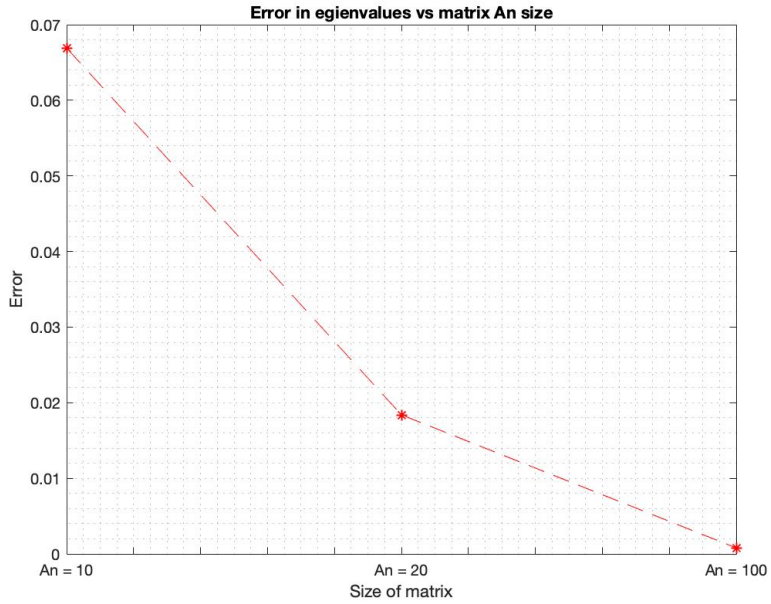

Fig. 9 The expansion of the equation in figure 6

V. Power Method

The power method is an algorithm that finds the extreme eigenvalues for a given matrix. The mechanics of the algorithm is iterative in nature. In the case of the normal power method algorithm, the largest eigenvalue is outputted. In order to find the smallest eigenvalue, a property of inverse matrices can be used. It is known that if the eigenvalues of A are λ, then the eigenvalues of A^{-1} are $\frac{1}{\lambda}$. This means that the inverse largest eigenvalue for A^{-1} is the smallest eigenvalue for A. Thus, the provided power method MATLAB script was modified to take the inverse of the inputted matrix and then the inverse was taken of the output value for the eigenvalue to generate the smallest eigenvalue.
To see the effectiveness of the algorithm, the A_{10}, A_{20}, and A_{100} matrices were inputted into the provided power method script, as well as the modified inverse power methods. The results were then compared against the results from the QR algorithm. The error plots are shown below:

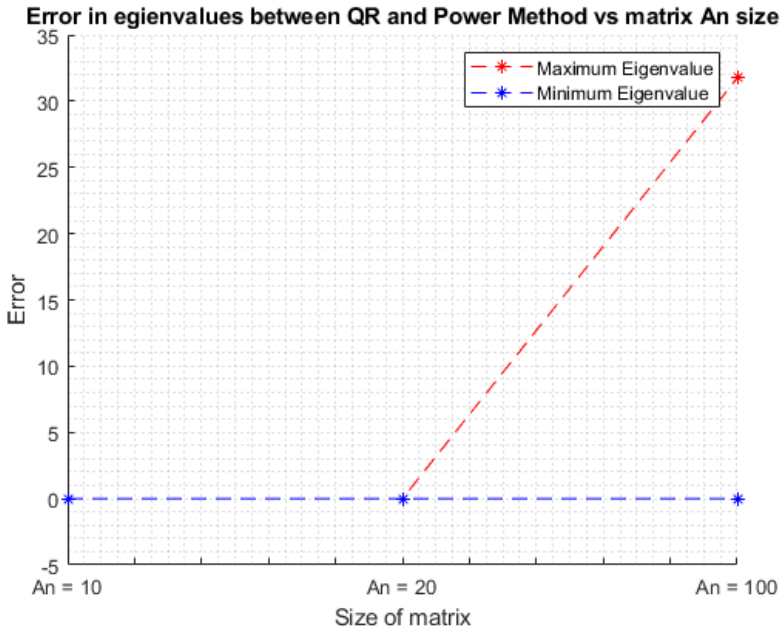

Fig. 10 The expansion of the equation in figure 6

Table 2 The error compared to actual eigenvalue for different sized matrices for the power method

An	n = 10	n = 20	n = 100
Error (Maximum)	1.14e-13	04.09e-12	31.74
Error (Minimum)	-1.78e-15	-1.78e-15	4.23-13

For the minimum eigenvalue (the inverse power method case), the error is close to or at zero for all three sized matrices. For the maximum eigenvalue (regular power method case), the error is inconsistent. For the A_{10} and A_{20} cases, rather than finding the maximum eignvalue, the power algorithm seems to find the second highest instead. However, for the A_{100} case, the eigenvalue found did not make sense at all. This is likely due to complications in the algorithm, and after significant troubleshooting we could not figure out why this was occurring.

VI. Conclusion

After using both the QR and the Power Method as a means of generating the eigenvalues for a matrix, A_n, it is clear that there are clear advantages and disadvantages to each method. The most significant limitation of the Power Method is that it can only be used to find the extreme (min/max) eigenvalues of the matrix. With that said, it is much easier computationally to setup the power method. To get all of the eigenvalues, the QR method must be used. The QR method is more computationally taxing and requires a great deal of coding to setup, but once that is done, it is definitely the preferred method.

VII. Appendix

A. Appendix A: MATLAB Code

Main script

```
%% this the main script to run the functions created, more info in
% Info folder.

clear
clc
close all

%% Get An

An_Matrix = An(3);
% the following is an example from a book to verfiy that the code is
% working, which is!
% An = [ 1 2 -1 ; 1 -1 2 ; 1 -1 2 ; -1 1 1 ];

% important note: due to computational accuracy, 0's aren't represented
% as 0's but rather very very small numbers (e-16 etc). so consider them
% 0

GrannShm = GS(An_Matrix);

% verfiy its' working:


% Verify set is orthogonal:

% any vector dotted with itself is == 1;
dot(GrannShm(:,1), GrannShm(:,1))
dot(GrannShm(:,2), GrannShm(:,2))
dot(GrannShm(:,3), GrannShm(:,3))

% any vector dotted with another one is == 0;
dot(GrannShm(:,1), GrannShm(:,2))
dot(GrannShm(:,1), GrannShm(:,3))
dot(GrannShm(:,2), GrannShm(:,3))


% Verify set is normal
% norm is == 1;

norm(GrannShm(:,1))
norm(GrannShm(:,2))
norm(GrannShm(:,3))

% use cgs.m to get QR Factorization:

[ Q1 R1 ] = cgs(An(3));
[ Q2 R2 ] = cgs(An(4));

% for An(3) i.e. An 3x3

A_3 = An(3);
Ak_3 = Q1*R1;
Ak1_3 = R1*Q1;

% prove they're equal
eigs(Ak_3)
eigs(Ak1_3)

% for An(4) i.e. An 4x4

A_4 = An(4);
Ak_4 = Q2*R2;
```

```
Ak1_4 = R2*Q2;

% prove they're equal
eigs(Ak_4)
eigs(Ak1_4)

%% apply it thousand times!!

%% An = 10
Matrix = {};

An10 = An(10);
Matrix{1} = An10;

for i = 1:1000


    [ Q R ] = cgs(Matrix{i});
    Matrix{i+1} = R*Q;

end

maximum = power_method(An10);
minimum = inverse_power_method(An10);

eigs =  diag(Matrix{end});

error_max10 = maximum - eigs(2)
error_min10 = minimum - min(diag(Matrix{end}))

An10_error = abs ( pi^2 - min(diag(Matrix{end})) )


%% An = 20
Matrix = {};

An20 = An(20);
Matrix{1} = An20;

for i = 1:1000


    [ Q R ] = cgs(Matrix{i});
    Matrix{i+1} = R*Q;

end

maximum = power_method(An20);
minimum = inverse_power_method(An20);

eigs =  diag(Matrix{end});

error_max20 = maximum - eigs(2)
error_min20 = minimum - min(diag(Matrix{end}))

An20_error = abs ( pi^2 - min(diag(Matrix{end})) )

%% An = 100
Matrix = {};

An100 = An(100);
Matrix{1} = An100;

for i = 1:1000


    [ Q R ] = cgs(Matrix{i});
    Matrix{i+1} = R*Q;
```

```
end
maximum = power_method(An100);
minimum = inverse_power_method(An100);

eigs =  diag(Matrix{end});

error_max100 = maximum - eigs(2)
error_min100 = minimum - min(diag(Matrix{end}))
An100_error = abs ( pi^2 - min(diag(Matrix{end})) )

%% plot error

figure(1)
plot([ 1 2 3 ],[ An10_error An20_error An100_error ],'--r*')
title('Error in egienvalues vs matrix An size')
xlabel('Size of matrix');
ylabel('Error');
xticklabels({'An = 10','','','','', 'An = 20','','','','', 'An = 100'})
grid minor

figure(2)
hold on
plot([ 1 2 3 ],[ error_max10 error_max20 error_max100 ],'--r*')
plot([ 1 2 3 ],[ error_min10 error_min20 error_min100 ],'--b*')
title('Error in egienvalues between QR and Power Method vs matrix An size')
xlabel('Size of matrix');
ylabel('Error');
xticklabels({'An = 10','','','','', 'An = 20','','','','', 'An = 100'})
legend("Maximum Eigenvalue","Minimum Eigenvalue")
grid minor
```

Creating *An*

```
function [ Matrix ] = An(n)
% this code will try to create the An matrix to estimate the eigenvalues
% for more information read section 3 in the folder info.
%
% - - - - - - - - - - - - - - - - - - -
%
% Inputs: n (size of nxn matrix)
%
% - - - - - - - - - - - - - - - - - - -
%
% Ouputs : Matrix nxn that has the pattren shown in section 3 in the
% folder.
%

% Bn is the matrix that has the pattrens -1 2 -1, see section 3.

Bn = zeros(n,n);

for i = 1:n

    if i == 1;
    Bn(i,i) = 2;
    Bn(i,i+1) = -1;

    elseif i == n

    Bn(i,i-1) = -1;
    Bn(i,i) = 2;


    else
```

```
    Bn(i,i-1) = -1;
    Bn(i,i) = 2;
    Bn(i,i+1) = -1;

    end

end


Matrix = (( n+1 / 1 ) ^2) * Bn ;

end
```

Gram-Schmidt

```
function [GS_matrix ] = GS(An)
% this function takes a linearly independent set of vectors
% and outputs an orthanormal set, using the study Gram-Schmidt
% process. In this function, inputs will be arranged in a matrix
% and the function will use the columns of the matrix to create
% an orthanormal set, and output them in matrix
%
% For more info, go into info file.
%
%
%

[ r c ] = size(An);


for i = 1:c

    % get summation term in GS.
    if i>1

    for j = 1:i

        if j == 1
        ortha(:,1) = An(:,i);

        else

       ortha(:,1) = ortha(:,1) - ( dot(GS_matrix(:,j-1), An(:,i))*GS_matrix(:,j-1)) ; ;


        end

    end

     GS_matrix(:,i) = ortha(:,1) / norm(ortha(:,1)) ;


    elseif i==1
        % this's the orthagonal vector, then we normalize.
        ortha(:,i) = An(:,i) ;
        %nromalize (so orthanormal).
        GS_matrix(:,i) = ortha(:,i) / norm(ortha(:,i)) ;

    end

end


end
```

```
function [lambda]=inverse_power_method(A)
A = inv(A);
tol=10^(-16);
s=size(A);
n=s(1);
z=ones(n,1);
lam=0;
for k=1:10000
    znew=A*z;
    lam_new=dot(A*znew,znew)/dot(znew,znew);
    if abs(lam_new-lam)<tol
        break
    end
    z=znew/norm(znew,2);
    lam=lam_new;
end
lambda=1/lam_new;
end
```