Analysis of Computing Eigenvalues
using a Modified Power Method and the QR Algorithm

By Adam Headley 998017147

Professor Cheer, MAT 128B – Numerical Analysis B, 16 March 2015

Consider a real symmetric matrix A, having distinct eigenvalues $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$. The inner product $< \mathbf{w}, \mathbf{z} >$ of the vectors $\mathbf{w} = [w_1, \ldots, w_n]^t$ and $\mathbf{z} = [z_1, \ldots, z_n]^t$, is defined by the dot product,

$$< \mathbf{w}, \mathbf{z} > = \sum_{i=1}^{n} w_i z_i.$$

Using a modified power method **mpm.M**, iterations of $\mathbf{v}^{(k)}$ are computed by the relation

$$\mathbf{v}^{(\mathrm{k})} = A\mathbf{v}^{(k-1)},$$

where $k = 1, 2, 3, \ldots$, starting with an initial vector $\mathbf{v}^{(0)} = [1,0,0,1,0,0,1,0,0,1]$. For each k the quantity

$$l_k = \frac{< \mathbf{v}^{(k)}, \mathbf{v}^{(k)} >}{< \mathbf{v}^{(k-1)}, \mathbf{v}^{(k)} >}.$$

In the $\lim_{k \to \infty} l_k$, it will be made clear that $l_k \to \lambda_1$, where λ_1 is the dominant eigenvalue of the real symmetric matrix A.

Every function that appears in this program was coded or default in MATLAB.

A The following will show, theoretically, that λ_1 is the dominant eigenvalue of the real symmetric matrix A and as well that $\mathbf{v}^{(\mathrm{k})}$ is the corresponding eigenvector, $\mathbf{x}_1$, of the matrix A.

Proof:

Note: $\mathbf{v}^{(0)} = \sum_{j=1}^{n} \alpha_j v_j$

$A\mathbf{v}^{(0)} = \sum_{j=1}^{n} \alpha_j A v_j = \sum_{j=1}^{n} \alpha_j \lambda_j v_j$

$$A^m\mathbf{v}^{(0)} = \sum_{j=1}^{n}\alpha_j A^m v_j = \sum_{j=1}^{n}\alpha_j\lambda_j^m v_j = \alpha_1\lambda_1^m v_1 + \lambda_1^m\sum_{j=2}^{n}\alpha_j\left(\frac{\lambda_j}{\lambda_1}\right)^m v_j$$

$$\mathbf{v}^{(\mathrm{k})} = \frac{A^m\mathbf{v}^{(0)}}{||A^m\mathbf{z}^{(0)}||} = \frac{\alpha_1\lambda_1^m v_1+\lambda_1^m\sum_{j=2}^{n}\alpha_j\left(\frac{\lambda_j}{\lambda_1}\right)^m v_j}{||\alpha_1\lambda_1^m v_1+\lambda_1^m\sum_{j=2}^{n}\alpha_j\left(\frac{\lambda_j}{\lambda_1}\right)^m v_j||} = \left(\frac{\lambda_1}{|\lambda_1|}\right)^k\frac{\alpha_1}{|\alpha_1|}v_1 + r_k$$

where $r_k \to 0$ as $k \to \infty$

$$l_k = \frac{<\mathbf{v}^{(\mathrm{k})},\mathbf{v}^{(\mathrm{k})}>}{<\mathbf{v}^{(\mathrm{k}-1)},\mathbf{v}^{(\mathrm{k})}>} = \frac{<\mathbf{v}^{(\mathrm{k})},\mathbf{v}^{(\mathrm{k}-1)}>}{<\mathbf{v}^{(\mathrm{k}-1)},\mathbf{v}^{(\mathrm{k}-1)}>} = \frac{<A\mathbf{v}^{(\mathrm{k}-1)},\mathbf{v}^{(\mathrm{k}-1)}>}{<\mathbf{v}^{(\mathrm{k}-1)},\mathbf{v}^{(\mathrm{k}-1)}>} \qquad \text{(See } \mathbf{mpm.M}\text{)}$$

$$= \frac{\sum_{j=1}^{n}|\alpha_j|^2\lambda_j^m v_j}{\sum_{j=1}^{n}|\alpha_j|^2\lambda_j^{m-1}v_j} = \lambda_1$$

For *stability* of the following methods, consider the *spectral radius* $\rho(A^{-1})$, or *largest eigenvalue* λ_1, of the matrix. If $\rho(A^{-1}) < 1$, then the matrix is stable. If $\rho(A) > 1$, then the matrix is unstable. All accounts of stability will be checked using the the algorithm built into **mpm.M**, line 36-41.

B The modified power method, **mpm.M**, was used to compute the dominant eigenvalue, $\lambda_1 = l_k$, and the corresponding normalized eigenvector, $\boldsymbol{v} = v_k$, for the matrices A and B given in problem #23 (Burden-Faires, 592),

$$A = \begin{bmatrix} 1+\alpha & -\frac{\alpha}{2} & 0 & \cdots & 0 \\ -\frac{\alpha}{2} & 1+\alpha & -\frac{\alpha}{2} & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & -\frac{\alpha}{2} \\ 0 & \cdots & 0 & -\frac{\alpha}{2} & 1+\alpha \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 1+\alpha & \frac{\alpha}{2} & 0 & \cdots & 0 \\ \frac{\alpha}{2} & 1+\alpha & \frac{\alpha}{2} & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \frac{\alpha}{2} \\ 0 & \cdots & 0 & \frac{\alpha}{2} & 1+\alpha \end{bmatrix}$$

with the stopping criteria, $|l_k - l_{\mathrm{k}-1}| \le 5E-6$. The matrices, A and B, used in this section are involved in the Crank-Nicolson method to solve the heat equation. The functions used to create them in the Appendix are labeled canonically **dluA.M** and **dluB.M**. The following table shows values for λ_k and $\mathbf{v}_k$ for corresponding values of α in matrices A and B:

The method was stable for all given values of α in the matrices A and B, however it was not stable for any given values of α in the matrix $A^{-1}B$.

Table 1: **MPM** Eigenvalues & Eigenvectors
corresponding to changing α in $A, B, A^{-1}B$

α	A		B		$A^{-1}B$	
	λ_1	$\boldsymbol{v}$	λ_1	$\boldsymbol{v}$	λ_1	$\boldsymbol{v}$
0.25	1.4603	$10^{12}\times$ -2.6815 4.5720 -5.0610 3.9245 -1.4764 -1.4764 3.9245 -5.0610 4.5720 -2.6815	1.4898	$10^{8}\times$ 2.9056 5.4693 7.4524 8.7589 9.3958 9.3958 8.7589 7.4524 5.4693 2.9056	1.4749	$10^{4}\times$ 1.4471 2.7553 3.8121 4.5449 4.9175 4.9175 4.5449 3.8121 2.7553 1.4471
0.5	1.9206	$10^{15}\times$ - 2.2031 3.7343 -4.1028 3.1602 -1.1842 -1.1842 3.1602 -4.1028 3.7343 -2.2031	1.9797	$10^{12}\times$ 0.7167 1.3609 1.8760 2.2288 2.4065 2.4065 2.2288 1.8760 1.3609 0.7167	1.9404	$10^{5}\times$ 1.3712 2.6225 3.6497 4.3749 4.7491 4.7491 4.3749 3.6497 2.6225 1.3712
0.75	2.3809	$10^{18}\times$ - 0.6858 1.1596 -1.2699 0.9752 -0.3648 -0.3648 0.9752 -1.2699 1.1596 -0.6858	2.4696	$10^{14}\times$ 2.5676 4.8901 6.7679 8.0711 8.7343 8.7343 8.0711 6.7679 4.8901 2.5676	2.3968	$10^{6}\times$ 0.3493 0.6689 0.9325 1.1194 1.2162 1.2162 1.1194 0.9325 0.6689 0.3493

C The modified power method, **mpm.M**, was used to compute the dominant eigenvalue, $\lambda_1 = l_k$, and the corresponding normalized eigenvector, $\boldsymbol{v} = v_k$, for the new matrix A given in problem #14 (Burden-Faires, 613),

$$A = \begin{bmatrix} 1-2\alpha & \alpha & 0 & \cdots & 0 \\ \alpha & 1-2\alpha & \alpha & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \alpha \\ \vdots & \ddots & \alpha & 1-2\alpha & \alpha \\ 0 & \cdots & 0 & \alpha & 1-2\alpha \end{bmatrix}$$

with the stopping criteria, $|l_k - l_{k-1}| \leq 5E-6$. This new matrix, A, created from the function **dluA14.M**, is involved in the Forward Difference method to solve the heat equation.

Table 2: **MPM** Eigenvalues & Eigenvectors corresponding to changes α in A

α		0.25	0.5	0.75
A	λ_1	0.9797	0.9595	-1.7619
	$\boldsymbol{v}$			$10^3\times$
		0.0779	0.0323	1.9886
		0.1485	0.0626	-3.3506
		0.2057	0.0869	3.6531
		0.2455	0.1052	-2.7938
		0.2658	0.1141	1.0428
		0.2658	0.1141	1.0428
		0.2455	0.1052	-2.7938
		0.2057	0.0869	3.6531
		0.1485	0.0626	-3.3506
		0.0779	0.0323	1.9886

The method was not stable for any given values of α in the new matrix A.

D This new matrix A, given in problem #14 (Burden-Faires, 613), is already symmetric and tri-diagonal. Using the QR algorithm, **QRa.M**, the computed eigenvalues of A, in problem #14 were obtained and then compared to the actual eigenvalues given in an equation in problem #15 (Burden-Faires, 613). These actual eigenvalues were computed using the function **eigA14.M**. The following table shows the differences between every actual eigenvalue and its corresponding computed eigenvalue (within the tolerance) in vector, d:

Table 2: **QRa** Difference between Actual Eigenvalues & Computed Eigenvalues corresponding to changes α in A

α		0.25	0.5	0.75
		$10^{-9}\times$	$10^{-11}\times$	
		0.380291909074515	4.712907841764036	-0.999999999883125
		-0.380292686230632	-4.132205688733848	-1.000000000116877
		-0.000000999200722	-0.000299760216648	-1.000000000000000
		0.000001998401444	-0.000099920072216	-0.999999999999997
A	d	-0.000003441691376	0	0.652711717831675
		-0.000000222044605	0	0.048408023804541
		0.000000333066907	-0.000099920072216	-1.836594758880455
		-0.000000055511151	-0.000299760216648	-1.000000000000000
		0.000000069388939	-4.132205688733848	-2.048408041836844
		0.000000006938894	4.712907841764036	-1.816116940918916

The differences were easy to compute for the first value of α, however the eigenvalues corresponding with the second value of α had zeros along the diagonal. After careful speculation I noticed the eigenvalues were strewn down the sub-diagonal and up the super-diagonal. After manually taking the difference, they ended up being the most accurate. For the third value of α something strange seems to be happening with the difference, almost exactly off by a whole number... very strange.

The method is unstable for all given values of α.

Appendix (7 functions)

```
function [ x ] = DP( w,z )
%DP Computes Dot Product
%   GIVEN: A: real symmetric matrix
%          Y = |y1| > |y2| > ... > |yn|: distinct eigenvalues of A
%   INPUT: w = [w1,...,wn]' vector
%              z = [z1,...,zn]' vector
%   OUTPUT: x = < w,z >: inner-product of vectors w,z

% dimension checks
while (size(w) ~= size(z))
    display('Vector dimensions must match')
    break;
end
% dimension assignment
if (size(w) == size(z))
    n = length(w);
end

% initialize sum
x = 0;

% dot product
for i = 1:n
    x = x + w(i)*z(i);
end

end
```

```
function [ D,L,U ] = dluA( n,a )
%dluA creates a sparse matrix and separates it into 3 parts
%    1. diagonal
%    2. lower triangular - diagonal
%    3. upper triangular - diagonal
```

```
tic
% diagonal
D = sparse(1:n,1:n,(1+a)*ones(1,n),n,n);

% S: subdiagonal;       S': superdiagonal
S = sparse(2:n,1:n-1,-(a/2)*ones(1,n-1),n,n);

% AD: lower antidiagonal;           AD': upper antidiagonal
% AD = sparse(n:-1:(n/2)+2,1:(n/2)-1,.5*ones(1,(n/2)-1),n,n);

% entries below the diagonal
L = S;

% entries above the diagonal
U = S';

M = D+L+U;

% Matrix Check
%M1 = D+S+S'+AD+AD';
%M1 = zeros(10)+M1;

toc
end
```

function [D,L,U] = dluB(n,a)

```
%dluB creates a sparse matrix and separates it into 3 parts
%    1. diagonal
%    2. lower triangular - diagonal
%    3. upper triangular - diagonal

tic
% diagonal
D = sparse(1:n,1:n,(1+a)*ones(1,n),n,n);

% S: subdiagonal;       S': superdiagonal
S = sparse(2:n,1:n-1,(a/2)*ones(1,n-1),n,n);
```

```
% AD: lower antidiagonal;          AD': upper antidiagonal
% AD = sparse(n:-1:(n/2)+2,1:(n/2)-1,.5*ones(1,(n/2)-1),n,n);

% entries below the diagonal
L = S;

% entries above the diagonal
U = S';

M = D+L+U;

% Matrix Check
%M1 = D+S+S'+AD+AD';
%M1 = zeros(10)+M1;

toc
end
```

```
function [ lk,yk,vk,k ] = mpm( n,a,v1,k,M )
%mpm Computes largest eigenvalue of matrix M iteratively
%    with left multiplication of v by powers of M
%   INPUT: n:  size of matrix A
%              a:  values in matrix A
%              v1: initial vector
%              k:  maximum # of iterations
%              M: Matrix A or B
%   OUTPUT: lk: computed eignevalue of M
%                 yk: eigenvalue of M check (eig(M))
%                 vk: corresponding eigenvector of yk
%                 k:  number of iterations

format long;

% creating nxn matrix
% Create matrix A
[DA,LA,UA] = dluA(n,a);
```

```
A = DA+LA+UA;

% Create matrix B
[DB,LB,UB] = dluB(n,a);
B = DB+LB+UB+zeros(n);

% Create matrix A14
[DA14,LA14,UA14] = dluA14(n,a);
A14 = DA14+LA14+UA14+zeros(n);

% Create matrix M
% M = A;
% M = B;
% M = A^(-1) * B;
M = A14;

% Stability of matrix M
if (max(abs(eig(M^-1))) <= 1)
    display('Method is stable \n');
elseif (max(abs(eig(M^-1))) > 1)
    display('Method is unstable \n');
end

% initialize first vector
vprev = v1;

% initialize quotients
l = zeros(k,1);
lrr = zeros(k,1); % Rayleigh-Ritz Quotient

% initialize first iteration
i = 1;

%MPM
while (i <= k)

    % next value of v
    vnext = M*vprev;
```

```
    % find l1 & move on
    if i == 1

      % Calculating l1
      % l(i) = dot(vnext,vnext)/dot(vprev,vnext);
      l(i) = DP(vnext,vnext)/DP(vprev,vnext);
      lrr(i) = DP(vnext,vprev)/DP(vprev,vprev); % Rayleigh-Ritz Quotient

      % prepare for next
      i = i+1;

      % pushing value
      vprev = vnext;

      % next value of v
      vnext = M*vprev;
    end

    % Calculating l
    % l(i) = dot(vnext,vnext)/dot(vprev,vnext);
    l(i) = DP(vnext,vnext)/DP(vprev,vnext);
    lrr(i) = DP(vnext,vprev)/DP(vprev,vprev); % =

    if norm(l(i) - l(i-1)) <= 5*10^-6;
        fprintf('The largest eigenvalue of A %s was found after %d iterations. \n',
double(l(i)),i);
        break;
    end

    % pushing value
    vprev = vnext;

    % next iteration
    i = i+1;
end

% computed eigenvalue
```

```
lk = l(i);
lrrk = lrr(i);

if (norm(lk - lrrk) <= 5*10^-6)
    display('Eigenvalue matches Rayleigh-Ritz Eigenvalue. \n');
end

%largest eigenvalue of l
yk = max(eig(M));

vk = vnext;

end
```

function [D,L,U] = dluA14(n,a)

```
%dluA14 creates a sparse matrix and separates it into 3 parts
%    1. diagonal
%    2. lower triangular - diagonal
%    3. upper triangular - diagonal

tic
% diagonal
D = sparse(1:n,1:n,(1-2*a)*ones(1,n),n,n);

% S: subdiagonal;       S': superdiagonal
S = sparse(2:n,1:n-1,a*ones(1,n-1),n,n);

% AD: lower antidiagonal;           AD': upper antidiagonal
% AD = sparse(n:-1:(n/2)+2,1:(n/2)-1,.5*ones(1,(n/2)-1),n,n);

% entries below the diagonal
L = S;

% entries above the diagonal
U = S';

M = D+L+U;
```

```
% Matrix Check
%M1 = D+S+S'+AD+AD';
%M1 = zeros(10)+M1;

toc
end
```

```
function [ Ak,y,d ] = QRa ( n,a,TOL )
%QRa uses function qr.M iteratively
%   INPUT: n: dimension length of A
%              a: values that determine the tridiagonal matrix A
%              TOL: tolerance (stopping criteria)
%   OUTPUT: Ak: Diagonal eigenvalue matrix
%                  y: actual eigenvalues from eigA14.M
%                  d: difference between the computed and actual eigenvalues

format short;

% Create matrix from problem #14
[D,L,U] = dluA14(n,a);
A1 = D+L+U;

% Stability of matrix A1
if (max(abs(eigs(A1^-1))) <= 1)
    display('Method is stable. \n');
elseif (max(abs(eigs(A1^-1))) > 1)
    display('Method is unstable. \n');
end

% Eigenvalue checker
y = eigA14(11,a);

% Initialized Q,R
[Q,R] = qr(A1);
```

```
Ak = R*Q;

% Error
error = norm(norm(y,inf) - norm(Ak,inf));

% QR loop
while error > TOL
    [Q,R] = qr(Ak);
    Ak = R*Q;
    error = norm(norm(y,inf) - norm(Ak,inf));
    if error < TOL
        Ak = diag(Ak+zeros(n));
        % difference
        d = y + Ak;
        return;
    end
end

end
```

```
function [ y ] = eigA14( m,a )
%eigA14 Computes the eigenvalues of matrix A
%       Described in problem #15 (Burden-Faires, 613)
%   INPUT: m: dimension+1
%              a: values that determine the tridiagonal matrix A
%   OUTPUT: y: vector of actual eigenvalues of A in Problem #14
%   The eigenvalues of the matrix A in Problem #14 are

y = zeros(m-1,1);
for i = 1:m-1
    y(i) = 1 - 4*a*(sin((pi*i)/(2*m)))^2;
end

end
```