Eigenvectors from Eigenvalues

Richard J Miller
http://www.urmt.org

A method to obtain the eigenvectors of a matrix with distinct eigenvalues by simple factorisation of the Cayley-Hamilton polynomial.

This is a formal write-up of a post I first made to sci.math 28/09/2019, corrected 01/10/2019.

This method does not require solving any algebraic equation and neither is this a numeric, iterative method. It is based upon the Cayley-Hamilton (CH) theorem and the associated CH polynomial.

Originality is unknown, but some references would be gratefully received - see further.

Matlab code is provided at the end, or email the author - see the web-site http://www.urmt.org for email.

This eigenvector method, originally known as the 'Residual Matrix Method' (RMM) in Unity Root Matrix Theory (URMT), was first published by the author in 2010 in the free, online PDF paper 'Pythagorean Triples as Eigenvectors and Related Invariants' available at:

http://www.urmt.org/pythag_eigenvectors_invariants.pdf

The motivation to post this Residual Matrix Method after ten or so years of it gathering dust was provided by a recent post on the science web site 'phys.org':

https://phys.org/news/2019-09-theorists-rosetta-stone-neutrino-physics.html

where the authors gave a new identity (in the eigenvalues) from which the eigenvectors can be found from an identity relation amongst the eigenvalues.

Their method was examined by Terence Tao et in his blog:

https://terrytao.wordpress.com/2019/08/13/eigenvectors-from-eigenvalues/

with an arXiv paper:

https://arXiv.org/abs/1908.03795

The Tao method also appears, like RMM, to require unique eigenvalues to avoid zero divisor problems. However, the method is restricted to Hermitian matrices, due to their construction from projection operators via the spectral theorem. Nevertheless, Hermitian limitations aside, the method is similar in aim to the RMM. The Tao method does not actually use the matrix itself, unlike RMM, and the identity is truly that among the eigenvalues only. However, it also requires the additional eigenvalues of an $(N-1)\times(N-1)$ (sub-matrix 'M' in the blog and Tao Paper), and I am not sure if the set of $N-1$ eigenvalues of M relate easily, if at all, to those already determined for the original matrix. It would not seem to be the case and this makes the method computationally intensive.

Back to my RMM (detailed further below), originality was not claimed at the time of publication (2010) given its seeming simplicity as regards the CH Theorem. Nevertheless, I would appreciate any references to it since I have not seen it in standard linear algebra textbooks, which either seem to go the standard equation solving route, or numerical determination. Some similarities to numerical methods were cited in the above paper, but this was vague at the time and remains vague now!

From a computational aspect, RMM does require N-1 matrix multiplications (without optimisation) to form an $N-1$ order matrix polynomial. Nevertheless, it is extremely simple and it does give both the column and row eigenvectors, without any conditions on the matrices such as symmetric or Hermitian.

Lastly, my original name for the method, i.e. 'Residual Matrix Method' is not the best! It was so named because, using the method, the eigenvectors are derived from the matrix remaining (termed the Residual Matrix) after the CH polynomial is factored by a single linear factor.

The Residual Matrix Method (RMM)

Defining

$\mathbf{A}$ = square, order N matrix with N unique eigenvalues,

$\mathbf{I}$ = $N \times N$ identity matrix

λ_k = kth unique eigenvalue, $k = 1 \ldots N$

V_i = ith eigenvector for eigenvalue λ_i

then the Cayley-Hamilton theorem basically says that **A** satisfies its own characteristic equation, i.e.

$$(\mathbf{A} - \lambda_1 \mathbf{I})(\mathbf{A} - \lambda_2 \mathbf{I}) \ldots (\mathbf{A} - \lambda_k \mathbf{I}) \ldots (\mathbf{A} - \lambda_N \mathbf{I}) = 0$$

written more concisely as the continued product

$$\prod_{k=1}^{N} (\mathbf{A} - \lambda_k \mathbf{I}) = 0$$

Assumption: Uniqueness of Eigenvalues

All eigenvalues λ_k, $k = 1 \ldots N$, are unique (or distinct) so that the CH polynomial is of full order N, i.e. contains a matrix term $\mathbf{A}^N$. If the eigenvalues are not all unique, i.e. one or more repeated, then the CH polynomial is of lesser order and not considered further. Note that this does not exclude zero eigenvalues, just repeated eigenvalues.

For example, in the 3x3 matrix case, the CH polynomial is

$$(\mathbf{A} - \lambda_1 \mathbf{I})(\mathbf{A} - \lambda_2 \mathbf{I})(\mathbf{A} - \lambda_3 \mathbf{I}) = 0$$

Explanation

Put very simply, by example, to get the first eigenvector V_1 for the first eigenvalue λ_1, multiply together all $N-1$ matrix factors $(\mathbf{A}-\lambda_2\mathbf{I})$, $(\mathbf{A}-\lambda_3\mathbf{I})\ldots(\mathbf{A}-\lambda_N\mathbf{I})$, omitting the factor $(\mathbf{A}-\lambda_1\mathbf{I})$ for the eigenvalue λ_1, to give a matrix $\mathbf{E}_1$, and then the eigenvector V_1 can be taken from any non-zero column of $\mathbf{E}_1$.

In more detail

To find the eigenvector V_1 for the ith eigenvalue λ_i, i.e.

$$\mathbf{A}V_i = \lambda_i V_i,$$

where all eigenvalues are unique (or distinct), i.e.

$$\lambda_i \neq \lambda_j \text{ for } i \neq j,\ i, j = 1\ldots N$$

First construct the 'residual matrix' $\mathbf{E}_i$ from the $N-1$ order polynomial formed from the continued product of the $N-1$ matrix factors $(\mathbf{A}-\lambda_k\mathbf{I})$ where $k \neq i$, i.e.

$$\mathbf{E}_i = \prod_{k=1,k\neq i}^{N}(\mathbf{A}-\lambda_k\mathbf{I}) = 0$$

In other words, the residual matrix $\mathbf{E}_i$ is just the CH polynomial excluding the factor $(\mathbf{A}-\lambda_i\mathbf{I})$, and is of degree $N-1$ with a highest term $\mathbf{A}^{N-1}$.

Thus, by the CH Theorem

$$(\mathbf{A}-\lambda_i\mathbf{I})\mathbf{E}_i = 0$$

Expanded and rearranged (using trivially $\mathbf{I}\mathbf{E}_i = \mathbf{E}_i$) gives

$$\mathbf{A}\mathbf{E}_i = \lambda_i\mathbf{E}_i$$

Since $\mathbf{E}_i$ is a matrix of N columns, this implies that each column must satisfy the eigenvector equation for eigenvalue λ_i, e.g. for column 1, $\mathbf{E}_i(:,1)$ in Matlab notation, then

$$\mathbf{A}\mathbf{E}_i(:,1) = \lambda_i\mathbf{E}_i(:,1)$$

Associating the eigenvector V_i to the first column of $\mathbf{E}_i$ (presuming it is not all zero), i.e.

$$V_i = \mathbf{E}_i(:,1)$$

then

$$\mathbf{A}V_i = \lambda_i V_i$$

and thus column one of $\mathbf{E}_i$ satisfies the eigenvector equation for eigenvalue λ_i, eigenvector V_i; likewise for all other non-zero columns of $\mathbf{E}_i$.

Since eigenvectors are not unique to within a non-zero scale factor, each column is actually a scalar multiple of the same eigenvector, but generally each scalar multiple is unique, albeit not necessarily so, and it may also be zero; neither is a limitation.

For example, for a 3×3 matrix $\mathbf{A}$ with eigenvector V_1, eigenvalue λ_1, then the 3x3 residual matrix $\mathbf{E}_1$ will have three columns, for some constants a,b,c given by

$$\mathbf{E}_1 = \begin{bmatrix} aV_1 & bV_1 & cV_1 \end{bmatrix}$$

Providing one of the constants a, b or c is not zero, then neither is the respective column zero, and any non-zero column gives a valid eigenvector V_1 of $\mathbf{A}$.

In fact, by defining a row vector V^1 as the three element vector

$$V^1 = \begin{bmatrix} a & b & c \end{bmatrix}$$

then $\mathbf{E}_1$ can be written as the outer product of column vector V_1 by row vector V^1 as in

$$\mathbf{E}_1 = V_1 V^1$$

Furthermore, row vector V^1 is actually a row eigenvector of $\mathbf{A}$ for the same eigenvalue λ_1, i.e.

$$V^1\mathbf{A} = \lambda_1 V^1$$

In fact, the scalar multiples, i.e. a,b,c in the above example, can be extracted by dividing one column by another non-zero column - see the Matlab code, vectors 'mV', at the end.

Notes

The eigenvalues have to all be unique because if one or more is repeated then the residual matrix **E** is all zero (see the subject of 'minimum polynomial' in Linear Algebra). Otherwise, since **E** is not all zero, at least one column is non-zero, i.e. one of a,b,c is not zero, in the example above.

This method works for real or complex-valued matrices and/or eigenvalues.

If the matrix is symmetric/Hermitian then the row eigenvectors are the transpose/Hermitian conjugate of the column eigenvectors.

A simple 2x2 Example

For a 2x2 matrix, with distinct eigenvalues λ_1 and λ_2, the CH polynomial is

$$(\mathbf{A}-\lambda_1\mathbf{I})(\mathbf{A}-\lambda_2\mathbf{I})=0,\ \lambda_1\neq\lambda_2$$

and the residual matrices $\mathbf{E}_1$ and $\mathbf{E}_2$ are simply the single factors

$$\mathbf{E}_1=(\mathbf{A}-\lambda_2\mathbf{I})$$

$$\mathbf{E}_2=(\mathbf{A}-\lambda_1\mathbf{I})$$

i.e. they are the same as the matrix **A** but with the alternate eigenvalue subtracted from the main diagonal:

$\mathbf{E}_1$ is formed by subtracting $\lambda_2\mathbf{I}$ from **A**

$\mathbf{E}_2$ is formed by subtracting $\lambda_1\mathbf{I}$ from **A**

Thus, no matrix multiplication is required and the eigenvectors V_1 and V_2 are simply read off from the columns of $\mathbf{E}_1$, $\mathbf{E}_2$

Eigenvector V_1 is obtained from the first column of $\mathbf{E}_1$ as

$$V_1=\begin{pmatrix}\mathbf{A}(1,1)-\lambda_2\\ \mathbf{A}(2,1)\end{pmatrix}$$

or it can be obtained from the second column as

$$V_1=\begin{pmatrix}\mathbf{A}(1,2)\\ \mathbf{A}(2,2)-\lambda_2\end{pmatrix}$$

Likewise, eigenvector V_2 can be obtained from the first column of $\mathbf{E}_2$ as

$$V_2=\begin{pmatrix}\mathbf{A}(1,1)-\lambda_1\\ \mathbf{A}(2,1)\end{pmatrix}$$

or it can be obtained from the second column as

$$V_2=\begin{pmatrix}\mathbf{A}(1,2)\\ \mathbf{A}(2,2)-\lambda_1\end{pmatrix}$$

In other words, you can read-off the eigenvectors from the matrix **A** after making a simple subtraction.

So, for example, the following matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 3 \\ 2 & 2 \end{pmatrix}$$

has the two distinct eigenvalues

$\lambda_1 = -1$ and $\lambda_2 = 4$

with CH polynomial

$$(\mathbf{A} + \mathbf{I})(\mathbf{A} - 4\mathbf{I}) = 0$$

After dividing this polynomial by factor $(\mathbf{A} + \mathbf{I})$ for eigenvalue $\lambda_1 = -1$, the residual matrix $\mathbf{E}_1$ for V_1 is

$$\mathbf{E}_1 = \begin{pmatrix} 1 & 3 \\ 2 & 2 \end{pmatrix} - 4\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} -3 & 3 \\ 2 & -2 \end{pmatrix}$$

The eigenvector V_1 is therefore

$$V_1 = \begin{pmatrix} -3 \\ 2 \end{pmatrix} \text{ or } V_1 = \begin{pmatrix} 3 \\ -2 \end{pmatrix}$$

where the second vector is thus -1 times the first and the row vector V^1 is thus

$$V^1 = \begin{pmatrix} 1 & -1 \end{pmatrix}.$$

For the eigenvector V_2 , the residual matrix $\mathbf{E}_2$ is

$$\mathbf{E}_2 = \begin{pmatrix} 1 & 3 \\ 2 & 2 \end{pmatrix} - (-1)\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ 2 & 3 \end{pmatrix}$$

so that

$$V_2 = \begin{pmatrix} 2 \\ 2 \end{pmatrix} \text{ or } V_2 = \begin{pmatrix} 3 \\ 3 \end{pmatrix},$$

where the second vector is thus 3/2 times the first and the row vector V^2 is thus [1 3/2].

$$V^2 = \begin{pmatrix} 1 & \frac{3}{2} \end{pmatrix}$$

Matlab Code

```
function residual_matrix_method
```

```
% residual_matrix_method.m (RMM)
%
% Determination of the eigenvectors for an NxN matrix
% when the N eigenvalues are known. Only eigenvectors for
% unique eigenvalues are determined, but not those for
% repeated eigenvalues. It can give both the column and
% row eigenvectors.
%
% This uses a method known as the 'residual matrix method' (RMM)
% in Unity Root Matrix Theory (URMT)
%
% The author published this method (*) in 2010 in the
% free, online PDF paper
%  'Pythagorean Triples as Eigenvectors and Related Invariants'
% available at:
%
%  http://www.urmt.org/pythag_eigenvectors_invariants.pdf
%
% See Section (4), page 6, Generation of Eigenvectors, which
% solves for three eigenvectors X+, X0 and X- from the
% known eigenvalues C,0,-C respectively for a 3x3 Unity
% Root Matrix A, as detailed in Sections (1) and (2).
%
% (*) Note that the author made no claim of originality
% at the time but neither could an exact reference be found,
% barring a similarity to the numerical Purification
% method, see the aforementioned paper for details.
%
% This method cannot determine the eigenvectors for repeated
% eigenvalues, but only those with unique eigenvalues.
%
% There are no restrictions that the matrix be symmetric
% or Hermitian which is only required for orthogonal
% eigenvectors (to unique eigenvalues).
%
% 28/09/2019 V1.0 R J Miller
%
% ************************************************
clear; % clear workspace variables
%%

% Firstly, demonstrate the Residual Matrix Method for an
% order N, random Hermitian matrix.
N=4; % arbitrary order N matrix, N>=2

% Construct an arbitrary, real-valued random matrix
A = rand(N,N);

% Alternative constructions of A follow, uncomment to suit

% Construct an arbitrary, complex-valued random matrix
% A = rand(N,N) + 1i*rand(N,N);

% construct a random Hermitian matrix
% tmp = rand(N,N); % generate a temporary random matrix
% A = (tmp + tmp')/2; % real-valued symmetric
% tmpi = 1i*rand(N,N); % imaginary random matrix
% A = A + (tmpi + tmpi')/2; % Hemitian


% ********************************************
```

```
% very simple 2x2 demonstrative case
% N=2;
% A = [1 3;2 2];

% % E1 = [-3 3;2 -2]
% % E2 = [2 3;2 3]

% % V1 = [3;-2] or [2;-2], L1 = -1
% % V2 = [2;2] or [3;3], L2 = 4

% ********************************************
% Extra matrix tests - use orthogonal eigenvectors
% and spectral decomposition to create test matrices
% for which all the eigenvectors are known (specified)
% and their eigenvector to be determined using RMM.

% N=4; % four orthogonal eigenvectors
X1=[1;1;1;1]/2;
X2=[1;-1;1;-1]/2;
X3=[-1;-1;1;1]/2;
X4=[-1;1;1;-1]/2;

% spectral decomposition - four unique eigenvalues 1,2,3,4
% A = X1*X1' + 2*X2*X2' + 3*X3*X3' + 4*X4*X4';

% two repeated zero eigenvalues for X1 and X2, RMM can
% only determine the eigenvectors X3 and X4 for non-zero
% eigenvalues 3 and 4 only
% A = 3*X3*X3' + 4*X4*X4';

% single zero eigenvalue OK for RMM
% A = 2*X2*X2' + 3*X3*X3' + 4*X4*X4';

% two repeated eigenvalue, '3' - problems with minimal
% polynomial
% A = X1*X1' + 2*X2*X2' + 3*X3*X3' + 3*X4*X4';

% three repeated eigenvalues, '3' - problems with minimal
% polynomial
% A = X1*X1' + 3*X2*X2' + 3*X3*X3' + 3*X4*X4';

% the above test matrices A are real-valued and symmetric
% by construction using X1,X2,X3,X4

% ********************************************
% Find eigenvectors and eigenvalues independently via
% Matlab - the eigenvalues are required as inputs to
% the Residual Matrix Method
[eigvecs,eigvals] = eigs(A);% Matlab eigenvectors,values

% The eigenvalues must be unique for this method and
% we should really check at this stage that the eigenvalues
% are all unique - but left unchecked here, assume
% uniqueness for now given A is a smallish random matrix
% or purposefully constructd for uniqueness (or
% non-uniqueness to test for errors

I = eye(N,N); % NxN identity matrix

% store of col eigenvectors determined by RMM
Vmstore = zeros(N,N);
```

```
% store of col eigenvalues determined by Matlab
Evastore = zeros(1,N);

mV = zeros(1,N); % mth row eigenvector of matrix A

% store of row eigenvectors determined by RMM
mVstore = zeros(N,N);

% Find eigenvector for mth eigenvalue, m=1..N
for mm=1:N

    % Initialise residual matrix for mth eigenvector
    Em = I;

    Lm = eigvals(mm,mm); % mth eigenvalue

    Evastore(mm) = Lm; % store eigenvalue for later

    % form residual matrix from N-1 factors of
    % Cayley-Hamilton matrix polynomial
    for nn=1:N
        % exclude mth eigenvalue from residual matrix,
        % hence there are only N-1 matrix factors in
        % the residual matrix
        if nn  ~= mm
            Em=Em*(A - eigvals(nn,nn)*I);
        end
    end

    % Check to see if residual matrix polynomial is not zero
    % which occurs for repeated eigenvalues
    if is_zero(Em, 1e-5)
        tmpstr = sprintf(...
        'Repeated eigenvalues, zero residual matrix m=%d',mm);
        warning(tmpstr);
        continue; % skip rest of loop - gives a zero eigenvector
    end

    % Each column of Em is a multiple of Vm, i.e.
    % each column of Em is an eigenvector Vm of A
    % scaled by a constant Ym where the Ym are generally
    % unique for each column, but no necessarily so,
    % and may be zero hence giving an all-zero in Em
    % hence an invalid, all zero eigenvector.
    % However, there will always be
    % at least one, non-zero column (*) and so the mth
    % eigenvector can always be found.
    % (*) The Ym actually form a row eigenvector
    % and neither are they zero.
    % The Ym can be found by the enties in one column by
    % anoher. For example, providing both the first and mth
    % column of Em, i.e. Em(:,1) and Em(:,m) are both not
    % zero then Em(:,1)./Em(:,m) is a constant vector
    % where every element is the same non-zero constant 'Ym'
    % as in Em(:,1)./Em(:,m) = Ym*ones(N,1).
    for kk=1:N

        % Vm = mth eigenvector from kth column of residual matrix
        Vm = Em(:,kk);
```

```
    % Check eigenvector equation
    % This check will fail if Vm is all zero (plausibly
    % possible) or the eigenvalue is repeated (and the
    % residual matrix is then zero) - this latter case
    % is an error, flagged as warning further above
    tmp = A*Vm - Lm*Vm; % check == 0
    if ~is_zero( tmp, 1e-5 )
        warning('A Eigenvector error %d',mm);
    end

    % if non-zero magnitude eigenvector, to within tolerance,
    % then store unit vector form for printing at end
    tmp = sqrt(Vm'*Vm); % eigenvector magnitude
    if tmp > 1e-5
        Vmstore(:,mm) = Vm/tmp;
    end;

end;

% At this point the mth column eigenvector has
% been determined and can now be used with the residual
% matrix to find the row eigenvectors.

% The row eigenvectors are known as 'conjugate' or
% 'reciprocal' eigenvectors in URMT - see the paper,
% Section (3) 'Divisibilty factors' for the
% vector components alpha,beta,gamma.
% The residual matrix Em is an outer product of the
% column eigenevector 'Vm' with its row eigenvector 'mV' to
% form the matrix projection operator Vm*mV.
% This below code uses this fact to determine the row eigenvector
% from the normalised column eigenvector and

for jj=1:N
    for kk=1:N

        % find first non-zero component of col eigenvector jj
        if abs(Vmstore(kk,mm)) > 1e-5

        % Note that Vmstore(:,mm) is the mth normalised col
        % eigenvector, and Vmstore(kk,mm) is thus the kth component.
        % Em(kk,jj) is the product of Vm(kk)*mV(jj) where
        % Vmstore(kk,mm) ~ Vm to within normalisation

            mV(jj) = Em(kk,jj)./Vmstore(kk,mm);

            break; % quit loop now, only needs one non-zero component

        end; % if

    end; % kk
end; % jj

% normalise row eigenvector to unity
tmp = sqrt(mV*mV'); % magnitude of row vector
if tmp > 1e-5
    mVstore(mm,:) = mV/tmp;
end

% Check row eigenvector equations
% This check will fail if mV is zero (plausibly
```

```
    % possible) or the eigenvalue is repeated and the
    % residual matrix is then zero - this latter case
    % is an error, flagged as warning further above
    tmp = mV*A - Lm*mV; % check == 0

    if ~is_zero( tmp, 1e-5 )
        warning('Row eigenvector error %d',mm);
    end

end; % mth eigenvector

% ***************************************************
% print column eigenvectors determined via RMM,
fout = 1;
for mm=1:N

    tmpstr = sprintf('L%d=%0.3f + %0.3fi\nVm(%d)=',...
        mm,real(Evastore(mm)),imag(Evastore(mm)),mm);
    print_nx1vector_cmplx( Vmstore(:,mm), tmpstr, fout, N )

    % final check on eigenvectors.
     Vm = Vmstore(:,mm);
     Lm = Evastore(mm);
     if is_zero( Vm, 1e-5 )
         warning('Eigenvector %d is zero, eigenvalue %0.3f'...
             ,mm,Lm); % repeated eigenvalue
     end
     tmp = A*Vm - Lm*Vm; % check == 0

%      if ~is_zero( tmp, 1e-5 )
%          warning('B Eigenvector error %d',mm);
%      end

end;

% for refrence, print out the eigenvectors determined via Matlab
fprintf('Matlab determined eigenvectors:\n');
for mm=1:N
    tmpstr = sprintf('L%d=%0.3f + %0.3fi\nVm(%d)=',...
        mm,real(eigvals(mm,mm)),imag(eigvals(mm,mm)),mm);
    print_nx1vector_cmplx( eigvecs(:,mm), tmpstr, fout, N )
end;

% Symmetric or Hermitian matrices only
% The row eigenvectors are orthogonal to the col
% eigenvectors for different eigenvalues.
% Thus, to within sign,
% the matrix product of the row eigenvectors with the col
% row vectors is the identity, i.e.
% tmp = abs(mVstore*Vmstore) - I;
% if ~is_zero( tmp, 1e-5 )
%     error('row eigenvectors not orthogonal to col eigenvectors');
% end

% return; % skip the rest for now

%%
% Now demonstrate the Residual Matrix Method as in the original
% paper, 'Pythagorean Triples as Eigenvectors', Appendix (C),
% see header above.
%
```

```
% This is for a 3x3 integer 'Hermitian-like' Unity Root Matrix
% with eigenvalues 1,0,-1 and two eigenvectors that are
% Pythagorean Triples (eigenvalues +/-1), and a third Hyperbolic
% Eigenvector

A = [0 -2 2;2 0 -1;2 -1 0]; % Unity Root Matrix - Pythagorean form
[eigvecs,eigvals] = eigs(A); % eigenvalues C = {-1,0,1}

% Matlab-computed eigenvectors 'eigvecs' not required further
% as they will be calculated.
I = eye(3,3);

% Calculate residual matrix Ep for eigenvalue C=1
Ep = (A - 0*I)*(A - -1*I); % written in full to show eigenvalues

% Ep = A*(A + I); % written as one might normally
C = 1; % eigenvalue = 1

Xp = Ep(:,1); % column 1 eigenvector

% check eigenvector for unity eigenvalue
if ~is_zero( A*Xp - C*Xp, 1e-5 )
    warning('A*Xp - C*Xp eigenvector error %d',1);
end

Xp = Ep(:,2); % column 2 eigenvector
% check eigenvector for unity eigenvalue
if ~is_zero( A*Xp - C*Xp, 1e-5 )
    warning('A*Xp - C*Xp eigenvector error %d',1);
end

Xp = Ep(:,3); % column 3 eigenvector
% check eigenvector for unity eigenvalue
if ~is_zero( A*Xp - C*Xp, 1e-5 )
    warning('A*Xp - C*Xp eigenvector error %d',1);
end

% Calculate residual matrix E0 for eigenvalue C=0
E0 = (A - 1*I)*(A - -1*I); % written in full to show eigenvalues
% E0 = (A - I)*(A + I); % written as one might normally

C = 0; % eigenvalue = 0

X0 = E0(:,1); % column 1 eigenvector for
% check eigenvector X0
if ~is_zero( A*X0 - C*X0, 1e-5 )
    warning('A*X0 - C*X0 eigenvector error %d',0);
end

X0 = E0(:,2); % column 2 eigenvector for eigenvalue C=0
% check eigenvector X0
if ~is_zero( A*X0 - C*X0, 1e-5 )
    warning('A*X0 - C*X0 eigenvector error %d',0);
end

X0 = E0(:,3); % column 3 eigenvector for eigenvalue C=0
% check eigenvector X0
if ~is_zero( A*X0 - C*X0, 1e-5 )
    warning('A*X0 - C*X0 eigenvector error %d',0);
end
```

```
% Calculate residual matrix Em for eigenvalue C=-1
Em = (A - 1*I)*(A - 0*I); % written in full to show eigenvalues
% Em = (A + I)*A; % written as one might normally

C = -1; % eigenvalue = -1

Xm = Em(:,1); % column 1 eigenvector for
% check eigenvector Xm
if ~is_zero( A*Xm - C*Xm, 1e-5 )
    warning('A*Xm - C*Xm eigenvector error %d',-1);
end

Xm = Em(:,2); % column 2 eigenvector for eigenvalue C=0
% check eigenvector Xm
if ~is_zero( A*Xm - C*Xm, 1e-5 )
    warning('A*Xm - C*Xm eigenvector error %d',-1);
end

Xm = Em(:,3); % column 3 eigenvector for eigenvalue C=0
% check eigenvector Xm
if ~is_zero( A*Xm - C*Xm, 1e-5 )
    warning('A*Xm - C*Xm eigenvector error %d',-1);
end

% The eigenvectors Xp,X0,Vm are not 'normalised to the DCE' in URMT
parlance
% Calculate reciprocal (row) eigenvectors using URMT 'conjugate relations'

T = [1 0 0;0 1 0;0 0 -1]; % Pythagorean T operator

X0 = X0/2; % convert to integer
Xm = Xm/-5; % convert to integer

pX = transpose(T*Xm);
mX = transpose(T*Xp);
oX = transpose(T*X0);

% check reciprocal (row) eigenvectors
% eigenvalue C=+1
if ~is_zero( pX*A - pX, 1e-5 )
    warning('pX*A - pX row eigenvector error %d',1);
end

if ~is_zero( oX*A, 1e-5 )
    warning('oX*A row eigenvector error %d',0);
end

% eigenvalue C=-1
if ~is_zero( mX*A + mX, 1e-5 )
    warning('mX*A + mX row eigenvector error %d',-1);
end

% invariant eigenvector inner product
if ~is_zero( pX*Xp - 2*C^2, 1e-5 )
    warning('pX*Xp inner product error %d',2);
end

% invariant zro eigenvector inner product - the DCE
if ~is_zero( oX*X0 - C^2, 1e-5 )
    warning('oX*X0 inner product error %d',1);
end
```

```
% invariant eigenvector inner product
if ~is_zero( mX*Xm - 2*C^2, 1e-5 )
    warning('mX*Xm inner product error %d',2);
end

% print the URM3 eigenvectors
N = 3;
print_nx1vector_cmplx( Xp, 'Xp=', fout, N )
print_nx1vector_cmplx( X0, 'X0=', fout, N )
print_nx1vector_cmplx( Xm, 'Xm=', fout, N )

return;

end

%%
% utility functions - not central to the method
function print_nx1vector_cmplx( V, start_msg, fout, nn )

error_tol = 1e-3;

% determine if any element is imaginary
imagf = any( (abs( imag(V(:)) ) > error_tol) > 0 );
fprintf(fout,'%s\n',start_msg);
if imagf > 0
    for ii=1:nn
        if abs( real( V(ii) ) ) > error_tol
            dp = abs( imag(V(ii)) - floor(imag(V(ii))) );
            if dp > error_tol
                fprintf(fout,'   %7.3f ',real(V(ii)) );
            else
                fprintf(fout,'   %7.3f ',real(V(ii)) );
            end
        else
            fprintf(fout,'       0 ');
        end
        if abs( imag( V(ii) ) ) > error_tol
            oneunit = abs( abs(imag(V(ii))) - 1 );
            if oneunit < error_tol
                if imag(V(ii)) > 0
                    fprintf(fout,'+      i,' );
                else
                    fprintf(fout,'+     -i,' );
                end
            else
                dp = abs( imag(V(ii)) - floor(imag(V(ii))) );
                if dp > error_tol
                    fprintf(fout,'+ %5.1fi,',imag(V(ii)) );
                else
                    fprintf(fout,'+ %5.0fi,',imag(V(ii)) );
                end
            end
        else
            fprintf(fout,'        ,');
        end
        fprintf(fout,'\n');

    end% ii

else
```

```
    % real-only, no imag component
    for ii=1:nn
        dp = abs( real(V(ii)) - floor(real(V(ii))) );
        if dp > error_tol
            fprintf(fout,'   %7.3f ',real(V(ii)) );
        else
            fprintf(fout,'   %7.3f ',real(V(ii)) );
        end
        fprintf(fout,'\n');
    end% ii
end

end

function rtn = is_zero(Xin, error_tol)
    rtn = 0;
    if norm(Xin) < error_tol
        rtn = -1;
    end
end
```