Covariances of Eigenvalues and Eigenvectors of a Symmetrix Matrix
Bill Menke, June 29, 2026

This note follows up upon "Research Note 9. Perturbation theory for algebraic eigenvalue problem", which dates from my graduate student days when a professor asked me how to calculate confidence limits of eigenvectors. Here, I extend my original formulas to the $N \times N$ case and test them using Monte Carlo simulations.

Perturbation theory for the eigenvalues and eigenvectors of a real symmetric matrix $\mathbf{A}$ is well-known, but I review it here for completeness. Suppose $\mathbf{A}_0$ is the unperturbed eigenvalue equation:

$$\mathbf{A}_0 \mathbf{v}_o^{(i)} = \lambda_0^{(i)} \mathbf{v}_o^{(i)} \quad \text{with} \quad \mathbf{v}_o^{(i)T} \mathbf{v}_o^{(j)} = \delta_{ij}$$

Perturbing $\mathbf{A}_0$ to $(\mathbf{A}_0 + \delta\mathbf{A})$ leads to corresponding perturbations in the eigenvalues and eigenverctors:

$$(\mathbf{A}_0 + \delta\mathbf{A}) \left(\mathbf{v}_o^{(i)} + \delta\mathbf{v}^{(i)} \right) = \left(\lambda_0^{(i)} + \delta\lambda^{(i)} \right) \left(\mathbf{v}_o^{(i)} + \delta\mathbf{v}^{(i)} \right)$$

Multiplying out, subtracting the unperturbed equation, and keeping terms to first order leads to

$$\delta\mathbf{A}\, \mathbf{v}_o^{(i)} + \mathbf{A}_0 \delta\mathbf{v}^{(i)} \approx \delta\lambda^{(i)} \mathbf{v}_o^{(i)} + \lambda_0^{(i)} \delta\mathbf{v}^{(i)}$$

As $\mathbf{v}_o^{(i)}$ is a unit vector, its perturbation must be normal to it:

$$\delta\mathbf{v}^{(i)} = \sum_{j \neq i} b_{ij} \mathbf{v}_o^{(j)}$$

Substituting and manipulating:

$$\delta\mathbf{A}\, \mathbf{v}_o^{(i)} + \mathbf{A}_0 \sum_{j \neq i} b_{ij} \mathbf{v}_o^{(j)} \approx \delta\lambda^{(i)} \mathbf{v}_o^{(i)} + \lambda_0^{(i)} \sum_{j \neq i} b_{ij} \mathbf{v}_o^{(j)}$$

$$\delta\mathbf{A}\, \mathbf{v}_o^{(i)} + \sum_{j \neq i} b_{ij} \mathbf{A}_0 \mathbf{v}_o^{(j)} \approx \delta\lambda^{(i)} \mathbf{v}_o^{(i)} + \lambda_0^{(i)} \sum_{j \neq i} b \mathbf{v}_o^{(j)}$$

Inserting the unperturbed equation $\mathbf{A}_0 \mathbf{v}_o^{(j)} = \lambda_0^{(j)} \mathbf{v}_o^{(j)}$ and rearranging leads to

$$\delta\mathbf{A}\, \mathbf{v}_o^{(i)} + \sum_{j \neq i} b_{ij} \lambda_0^{(j)} \mathbf{v}_o^{(j)} \approx \delta\lambda^{(i)} \mathbf{v}_o^{(i)} + \lambda_0^{(i)} \sum_{j \neq i} b_{ij} \mathbf{v}_o^{(j)}$$

$$\delta\mathbf{A}\, \mathbf{v}_o^{(i)} \approx \delta\lambda^{(i)} \mathbf{v}_o^{(i)} + \sum_{j \neq i} \left(\lambda_0^{(i)} - \lambda_0^{(j)} \right) b_{ij} \mathbf{v}_o^{(j)}$$

Multiplying by $\mathbf{v}_o^{(i)T}$ and applying normality $\mathbf{v}_o^{(i)T} \mathbf{v}_o^{(i)} = 1$ yields a formula for the perturbations to the eigenvalues:

$$\mathbf{v}_o^{(i)T} \delta\mathbf{A}\, \mathbf{v}_o^{(i)} \approx \delta\lambda^{(i)} \mathbf{v}_o^{(i)T} \mathbf{v}_o^{(i)} + \sum_{j \neq i} b_{ij} \left(\lambda_0^{(i)} - \lambda_0^{(j)} \right) \mathbf{v}_o^{(i)T} \mathbf{v}_o^{(j)}$$

$$\delta\lambda^{(i)} \approx f^{(ii)} \quad \text{with} \quad f^{(ii)} \equiv \mathbf{v}_o^{(i)T} \delta\mathbf{A}\, \mathbf{v}_o^{(i)}$$

Multiplying by $\mathbf{v}_o^{(k)T}$, for $k \neq i$ and applying orthogonality $\mathbf{v}_o^{(k)T}\mathbf{v}_o^{(i)} = 0$ yields a formula for the eigenvectors:

$$\text{for } k \neq i \quad \mathbf{v}_o^{(k)T}\delta\mathbf{A}\,\mathbf{v}_o^{(i)} \approx \delta\lambda^{(i)}\mathbf{v}_o^{(k)T}\mathbf{v}_o^{(i)} + \left(\lambda_0^{(i)} - \lambda_0^{(j)}\right)\sum_{j\neq i} b_{ij}\mathbf{v}_o^{(k)T}\mathbf{v}_o^{(j)}$$

$$\mathbf{v}_o^{(k)T}\delta\mathbf{A}\,\mathbf{v}_o^{(i)} \approx \sum_{j\neq i}\left(\lambda_0^{(i)} - \lambda_0^{(j)}\right) b_{ij}\delta_{jk} = \left(\lambda_0^{(i)} - \lambda_0^{(k)}\right) b_{ik}$$

$$\delta\mathbf{v}^{(i)} = \sum_{j\neq i} b_{ij}\mathbf{v}_o^{(j)} \quad \text{with} \quad b_{ij} \approx \frac{f^{(ij)}}{\lambda_0^{(i)} - \lambda_0^{(j)}} \quad \text{and} \quad f^{(ij)} \equiv \mathbf{v}_o^{(j)T}\delta\mathbf{A}\,\mathbf{v}_o^{(i)}$$

Now let us use this formula to calculate covariances, relying on the well-known rules: if $\delta\mathbf{y} = \mathbf{M}\delta\mathbf{x}$ then $\text{cov}(\delta\mathbf{y}) = \mathbf{M}\,\text{cov}(\delta\mathbf{x})\,\mathbf{M}^T$ and if $\mathbf{x} = \mathbf{x}_0 + \delta\mathbf{x}$ then $\text{cov}(\mathbf{x}) = \text{cov}(\delta\mathbf{x})$ (as $\mathbf{x}_0$ is a reference vector with zero covariance). Suppose the unique (upper triangular) components of $\delta\mathbf{A}$ are written as the vector $\delta\mathbf{a}$, e.g. in the 3×3 Voigt ordering case as:

$$\delta\mathbf{a} = [\delta\mathbf{A}_{11}, \delta\mathbf{A}_{22}, \delta\mathbf{A}_{33}, \delta\mathbf{A}_{23}, \delta\mathbf{A}_{13}, \delta\mathbf{A}_{12}\,]^T$$

or the 3×3 row-wise ordering as:

$$\delta\mathbf{a} = [\delta\mathbf{A}_{11}, \delta\mathbf{A}_{12}, \delta\mathbf{A}_{13}, \delta\mathbf{A}_{22}, \delta\mathbf{A}_{23}, \delta\mathbf{A}_{33}\,]^T$$

so that the quadratic from $f^{(ij)} \equiv \mathbf{v}_o^{(i)T}\delta\mathbf{A}\,\mathbf{v}_o^{(j)}$ becomes $f^{(ij)} \equiv \mathbf{u}_0^{(ij)T}\,\delta\mathbf{a}$ with, e.g. for the 3×3 Voigt case as:

$$\mathbf{u}_0^{(ij)} = \left[\mathbf{v}_{o1}^{(i)}\mathbf{v}_{o1}^{(j)}, \mathbf{v}_{o2}^{(i)}\mathbf{v}_{o2}^{(j)}, \mathbf{v}_{o3}^{(i)}\mathbf{v}_{o3}^{(j)}, \mathbf{v}_{23}^{(i)}\mathbf{v}_{32}^{(j)} + \mathbf{v}_{32}^{(i)}\mathbf{v}_{23}^{(j)}, \mathbf{v}_{13}^{(i)}\mathbf{v}_{31}^{(j)} + \mathbf{v}_{31}^{(i)}\mathbf{v}_{13}^{(j)}, \mathbf{v}_{12}^{(i)}\mathbf{v}_{21}^{(j)} + \mathbf{v}_{21}^{(i)}\mathbf{v}_{12}^{(j)}\right]^T$$

Then, as $\delta\lambda^{(i)} \approx f^{(ii)}$, we can write

$$\delta\boldsymbol{\lambda} \equiv \begin{bmatrix}\delta\lambda^{(1)}\\ \delta\lambda^{(2)}\\ \delta\lambda^{(3)}\end{bmatrix} = \begin{bmatrix}f^{(11)}\\ f^{(22)}\\ f^{(33)}\end{bmatrix} = \begin{bmatrix}\mathbf{u}_0^{(11)T}\\ \mathbf{u}_0^{(22)T}\\ \mathbf{u}_0^{(33)T}\end{bmatrix}\delta\mathbf{a} \equiv \mathbf{U}_0\delta\mathbf{a}$$

And as

$$\delta\mathbf{v}^{(i)} = \sum_{j\neq i}\frac{\mathbf{u}_0^{(ij)T}\,\delta\mathbf{a}}{\lambda_0^{(i)} - \lambda_0^{(j)}}\mathbf{v}_o^{(j)} = \left\{\sum_{j\neq i}\left[\frac{\mathbf{v}_o^{(j)}\mathbf{u}_0^{(ij)T}}{\lambda_0^{(i)} - \lambda_0^{(j)}}\right]\right\}\delta\mathbf{a} \equiv \mathbf{W}_0^{(i)}\delta\mathbf{a}$$

we can write

$$\delta\mathbf{V} \equiv \begin{bmatrix}\delta\mathbf{v}^{(1)}\\ \delta\mathbf{v}^{(2)}\\ \delta\mathbf{v}^{(3)}\end{bmatrix} = \begin{bmatrix}\mathbf{W}_0^{(1)}\\ \mathbf{W}_0^{(2)}\\ \mathbf{W}_0^{(3)}\end{bmatrix} \equiv \mathbf{W}_0\delta\mathbf{a}$$

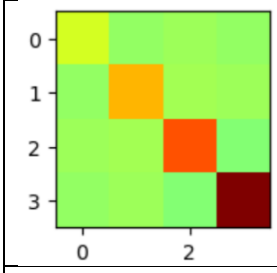
Then, if $\delta\mathbf{a}$ has covariance $\text{cov}(\delta\mathbf{a})$, by the error propagation:

$$\text{cov}(\boldsymbol{\lambda}) = \text{cov}(\delta\boldsymbol{\lambda}) = \mathbf{U}_0\,\text{cov}(\delta\mathbf{a})\;\mathbf{U}_0^T$$

$$\mathrm{cov}(\mathbf{V}) = \mathrm{cov}(\delta\mathbf{V}) = \mathbf{W}_0\,\mathrm{cov}(\delta\mathbf{a})\;\mathbf{W}_0^T$$

$$\mathrm{cov}(\boldsymbol{\lambda}, \mathbf{V}) = \mathrm{cov}(\delta\boldsymbol{\lambda}, \delta\mathbf{V}) = \mathbf{U}_0\,\mathrm{cov}(\delta\mathbf{a})\;\mathbf{W}_0^T$$

The following example is for a 4×4 matrix and uses row-wise ordering. The calculated covariances compare well to those calculated from 10,000 realizations of $\delta\mathbf{A}$. Such realizations can be computed using the rule $\delta\mathbf{a} = [\mathrm{cov}(\delta\mathbf{a})]^{½}\mathbf{z}$, where $\mathbf{z}$ is a Normally-distributed variable with zero mean and unit variance. Here $\mathbf{M}^{½}$ is the symmetric square root of $\mathbf{M}$.

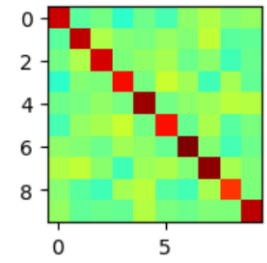


An exemplary $\mathbf{A}_0$

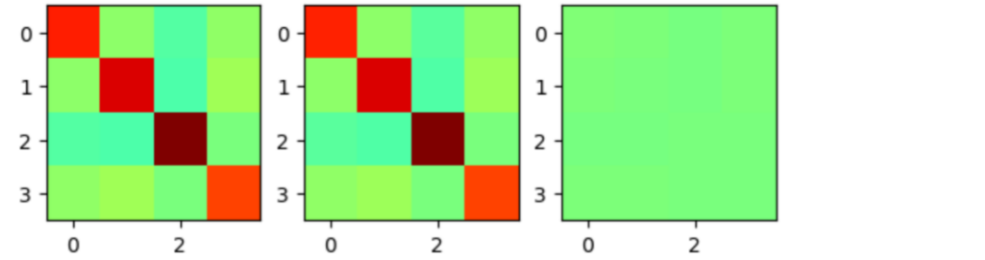
```
lam0:
[[0.9789311 ]
 [1.98155948]
 [3.53440942]
 [5.20509999]]

v0:
[[ 0.97558706 -0.10433272  0.16891664  0.09387092]
 [-0.16990317 -0.9158071   0.32779126  0.15806064]
 [-0.12931292  0.37018966  0.91301299  0.11245031]
 [-0.05139044  0.1156359  -0.17443436  0.97650398]]
```

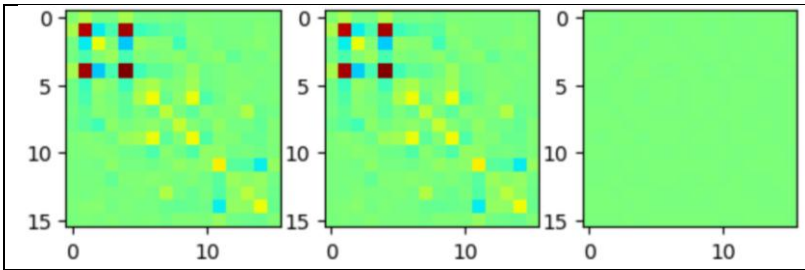
$\boldsymbol{\lambda}_0$ and $\mathbf{v}_0^{(i)}$



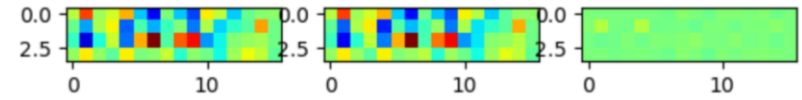
An exemplary $\mathrm{cov}(\delta\mathbf{a})$



Perturbative $\mathrm{cov}(\boldsymbol{\lambda})$ (left), Monte Carlo $\mathrm{cov}(\boldsymbol{\lambda})$ (middle) and difference (right)



Perturbative $\mathrm{cov}(\mathbf{V})$ (left), Monte Carlo $\mathrm{cov}(\mathbf{V})$ (middle) and difference (right)



Perturbative $\mathrm{cov}(\boldsymbol{\lambda}, \mathbf{V})$ (left), Monte Carlo $\mathrm{cov}(\boldsymbol{\lambda}, \mathbf{V})$ (middle) and difference (right)

Python code for the $N \times N$ case:

```
import numpy as np;
from scipy import linalg as la;
from math import sqrt;
from matplotlib import colormaps;
from matplotlib import pyplot as plt;

def index_arrays(N):
    M = int(N*(N+1)/2);
    i_of_k = np.zeros((M,1), dtype=int );
    j_of_k = np.zeros((M,1), dtype=int );
    k_of_ij = np.zeros((N,N), dtype=int );
    k=0;
    for i in range(0,N):
        for j in range(i,N):
            i_of_k[k,0] = i;
            j_of_k[k,0] = j;
            k_of_ij[i,j]=k;
            k_of_ij[j,i]=k;
            k=k+1;
    return i_of_k, j_of_k, k_of_ij;
def symmatrix( A ):
    # copies upper triangle of symmetric matrix to lower triangle
    N,i = np.shape(A);
    for i in range(0,N-1):
        for j in range(i+1,N):
            A[j,i]=A[i,j];
    return(A);
def tovector( A ):
    # vector v from a symmetric NxN matrix A
    N,i = np.shape(A);
    i_of_k, j_of_k, k_of_ij = index_arrays(N);
    M,i = np.shape(i_of_k);
    a = np.zeros((M,1));
    for k in range(M):
        i = i_of_k[k,0];
        j = j_of_k[k,0];
        print( i,j,k,N,M );
        a[k,0] = A[i,j];
```

```
    return(a);
def fromvector( a ):
    # NxN symmetric matrix A from Mx1 vector representation a
    # M = N (N+1) / 2;   2M = N**2 + N;  N**2 + N - 2M = 0
    # A=1, B=1, C=-2M
    # N = ( -B +/- sqrt(B*2 - 4AC ) ) / 2A
    # N = ( -1 + sqrt( 1 + 8M ) ) / 2
    M, i = np.shape(a);
    N = int( 0.01 + ( -1 + sqrt( 1 + 8*M ) ) / 2 );
    i_of_k, j_of_k, k_of_ij = index_arrays(N);
    A = np.zeros((N,N));
    for k in range(M):
        i = i_of_k[k,0];
        j = j_of_k[k,0];
        A[i,j]=a[k,0];
    A=symmatrix(A);
    return(A);
def do_u0(v0,i,j):
    # quadratic form F = v(i).T dA v(j)
    # written in so that F = u0(i,j) da
    # note that dlam(i) = F(i,i) = u0(ii).T da
    N, tmp = np.shape(v0);
    i_of_k, j_of_k, k_of_ij = index_arrays(N);
    M = int( N*(N+1)/2 );
    u0 = np.zeros((M,1));
    for p in range(N):
        for q in range(N):
            k = k_of_ij[p,q];
            u0[k,0] = u0[k,0] + v0[p,i]*v0[q,j];
    return(u0);
def do_W0i(i, v0, lam0):
    # eigenvector perturbations as a linear function of da
    # W0i such that dvi = W0i da
    N, tmp = np.shape(lam0);
    M = int(N*(N+1)/2);
    W0i = np.zeros((N,M));
    for j in range(N):
        if i==j:
            continue;
        W0i=W0i+np.matmul(v0[0:N,j:j+1],do_u0(v0,i,j).T)/(lam0[i,0]-lam0[j,0]);
    return(W0i);
def eigcovN( a0, cova ):
# covariace of eigenvalues and eigenvectors of 3x3 real symmeteric matrix A0
# represented as 6x1 Voigt vector
# a0 = [ A0[0,0], A0[1,1], A0[2,2]], A0[1,2]], A0[0,2], A0[0,1] ].T
# so that cova is 6x6.
# Returned:
# 3x3 covlam: covariance of eigenvalues
# 9x9 covv: covariance of eigenvectors, arranged v[0,0], v[1,0], v[2,0], v[1,0], ...
# 3x9 covlamv: covariance of eigenvalues with eigenvectors
    A0 = fromvector(a0);
    N,i = np.shape(A0);
    M,i = np.shape(a0)
    lam0, v0 = la.eigh(A0);
    lam0 = np.reshape(lam0,(N,1));
    U0 = np.zeros((0,M));
    for i in range(N):
        U0 = np.concatenate( (U0,do_u0(v0,i,i).T), axis=0 );
```

```
    W0 = np.zeros((0,M));
    for i in range(N):
        W0 = np.concatenate( (W0, do_W0i(i,v0,lam0)), axis=0 );
    covlam = np.matmul( U0, np.matmul( cova, U0.T ) );
    covv = np.matmul( W0, np.matmul( cova, W0.T ) );
    covlamv = np.matmul( U0, np.matmul( cova, W0.T ) );
    return covlam, covv, covlamv;

N=4;
M=int( N*(N+1)/2 );
if( N==3 ):
    a0=np.array([[1.1],[0.3],[0.2],[2.2],[-0.2],[3.3]]);
elif( N==4 ):
    a0=np.array([[1.1],[0.3],[0.4],[0.3],[2.2],[0.5],[0.4],[3.3],[0.1],[5.1]]);
A0 = fromvector(a0);
lam0, v0 = la.eigh(A0);
lam0 = np.reshape(lam0,(N,1));

cova=0.001*np.identity(M)+symmatrix( np.random.normal(
loc=0.0,scale=0.0001,size=(M,M) ) );
sqrt_cova = la.sqrtm(cova);
covlam, covv, covlamv = eigcovN( a0, cova );
```