

# Iterative subspace decomposition

**Source:** <http://sci.tech-archive.net/Archive/sci.math.num-analysis/2005-02/0005.html>

---

**From:** Ryan Mitchley ([ryanmit\\_at\\_worldonline.co.za](mailto:ryanmit_at_worldonline.co.za))

**Date:** 01/31/05

Date: 31 Jan 2005 02:32:42 -0800

Hi all

Sorry for the long post. This is probably a question for the fairly brave .

..

I need to perform iterative subspace decomposition of data acquired from a sensor array. At the moment I am using MATLAB to simulate the system. So far, I have successfully implemented the PASTd algorithm (based on this document: <http://www.astron.nl/~hampson/adapalg.pdf> although I think the technique was originally outlined by B. Yang, in "Projection Approximation Subspace Tracking," IEEE Transactions on Signal Processing, vol. 43, pp. 95-107, January 1995.). MATLAB code is below.

Unfortunately, the PASTd algorithm does not produce orthogonal eigenvectors. I have found a couple of iterative algorithms that do, including one called NFRQ, described here: <http://citeseer.ist.psu.edu/683342.html>. Although the algorithm is described quite nicely on page 3, I am struggling to get it to work in MATLAB. Basically, although the implementation seems to be syntactically correct, it is not producing the eigendecomposition I expect at all.

In the hopes that someone can unravel my problem, I have posted the MATLAB code below. It seems to be a fairly useful algorithm, so I hope that that is something of a reward to anyone who can help!

Thanks for any and all replies!

Ryan Mitchley

```

function TestNFRQ;
% TestNFRQ.m

M = 5; % 5 sensors in array
N = 4000; % 4000 snapshots in time

% Create a synthetic correlation matrix
Rxx = zeros(M, M);
for a = 1:1:M
for b = 1:1:M
if (a <= b)
Rxx(a, b) = a/b;
else
Rxx(a, b) = b/a;
end
end
end
Rxx
% Factorise the synthetic correlation matrix
A = chol(Rxx)';
% Create random data X
X = (1/sqrt(2)).*(randn(M,N) + j*randn(M,N));
MeanX = sum(sum(X)) / (M*N);
X = X - MeanX; % ensure X is zero mean
InvV = 1./ sqrt(var(X'));
for m = 1:1:M
X(m,:) = X(m,:) .* InvV(m); % ensure variance of each row is 1.0
end;
Y = A*X; % "colour" the random data by the factorised correlation
matrix
Rest = (1/N) .* Y*Y' % form an estimate of the correlation matrix

% Use MATLAB's built-in 'eig' function, then sort by descending order
of eigenvalue
[EvectorUnsorted, EvaluesEig] = eig(Rest);
[EvaluesEig,i]=sort(diag(EvaluesEig)');
i = fliplr(i);
for n = 1:1:M
EvectorEig(:,n) = EvectorUnsorted(:,i(n));
end
EvaluesEig = diag(fliplr(EvaluesEig));

EvaluesEig
EvectorEig
Reig = EvectorEig * EvaluesEig * EvectorEig' % synthesize the

```

## sci.math.num-analysis: Iterative subspace decomposition

correlation matrix from its eigendecompositions

```
% SVD Approach
% [EvectorSVD EvaluesSquaredSVD ErightSVD] = svd(Y, 0); % Calculate
eigendecomposition using SVD of the data matrix
% EvaluesSVD = (1/N) .* diag(diag(EvaluesSquaredSVD).^2);
% EvaluesSVD
% EvectorSVD
% Rsvd = EvectorSVD * EvaluesSVD * EvectorSVD' % synthesize the
correlation matrix from its eigendecompositions
```

```
% Iterative approach
Wpd = eye(M); % initialise eigenvectors to identity matrix
Cpd = eye(M); % initialise matrix of eigenvalues on the main
diagonal
Wnf = Wpd;
Cnf = Cpd
Beta = 0.999;
```

```
% Divide the simulated data into timeslices and "feed" to various
algorithms
```

```
for t = 1:1:N
v = Y(:,t); % get the next set of sensor samples
[Wpd, Cpd] = ePASTd(v, Wpd, Cpd, Beta, M); % call PASTd
iteration
[Wnf, Cnf] = eNFRQ(v, Wnf, Cnf, Beta); % call NFRQ
iteration
% This one isn't working
end;
Cpd = Cpd .* ((1 - Beta) / Beta); % Scale eigenvalues because of
processing gain due to Beta
Wpd
Cpd
Rpd = Wpd * Cpd * Wpd' % synthesize the correlation matrix
from its eigendecompositions
Wnf
Cnf
Rnf = Wnf * Cnf * Wnf' % synthesize the correlation matrix
from its eigendecompositions
```

```
%-----
function [W, C] = ePASTd(v, InW, InC, Beta, M);
```

```
% Simulate the PASTd algorithm for iterative eigendecomposition
% (Projection Approximation Subspace Tracking with Deflation)
%
% v : input array snapshot, corresponding to the most recent instant
in time
% InW : input matrix containing eigenvectors as columns
% InC : matrix containing the corresponding eigenvalues on the main
diagonal
```

## sci.math.num-analysis: Iterative subspace decomposition

```
% Beta : forgetting factor
% M : dimension of (principal) subspace to be calculated
% W : output matrix containing eigenvectors as columns
% C : output matrix containing the corresponding eigenvalues on the
main diagonal
```

```
c = diag(InC);
W = InW;
for j = 1:1:M % calculate all M eigenvalues/eigenvectors
x = (W(:,j))*v;
c(j) = Beta * c(j) + abs(x)^2;
e = v - W(:,j)*x;
W(:,j) = W(:,j) + e*(x'/c(j));
v = v - W(:,j)*x;
end;
C = diag(c); % return eigenvalues as a diagonal matrix
```

```
%-----
function [OutW, OutC] = eNFRQ(r, InW, InC, Beta);
```

```
% Simulate the NFRQ algorithm for iterative eigendecomposition.
% This is the algorithm that isn't working.
%
% r : input array snapshot, corresponding to the most recent instant
in time
% InW : input matrix containing eigenvectors as columns
% InC : matrix containing the corresponding eigenvalues on the main
diagonal
% Beta : forgetting factor
% W : output matrix containing eigenvectors as columns
% C : output matrix containing the corresponding eigenvalues on the
main diagonal
```

```
Gamma = 0.0000001;
```

```
W = InW;
p = diag(InC)';
```

```
y = W' * r;
BetaSopt = Beta / (2 * (norm(r).^2) + Gamma);
Rho = 4 * BetaSopt * (1 + BetaSopt .* (norm(r).^2));
Tau = (1 / (norm(y).^2)) * (1 / sqrt(1 + Rho*(norm(y).^2)) - 1);
p = (Tau * (W * y)) / BetaSopt + 2 * r * (1 + Tau*(norm(y).^2));
OutW = W + BetaSopt * p * y';
OutC = diag(p);
```