AMATH 482 Homework 5

Rishabh Verma

March 17th, 2021

Abstract

Sequential frames of a video can be regarded as a transformation of the previous. In the process of Dynamic Mode Decomposition, this transformation is assumed to be linear and decomposed. Out of the resulting components, the constant-order ones can be used to isolate the background of the video. The background is then subtracted from the original video to successfully isolate the foreground.

1 Introduction and Overview

Background subtraction is the process of isolating the foreground and background from a video. The task of this lab is to carry out background separation on two videos, both of which are filmed from a fairly stable camera perspective.

Both videos have a resolution of 540x960 and are 6 seconds long. They are both screen-recordings of a device displaying the video, and so are re-sampled to be 60 frames per second, though the fact that each frame is accompanied by two or three duplicate frames suggests that the original source material is closer to 24 frames per second. All duplicate frames are later stripped anyway.

The subject of the first video is a group of racecars on a racetrack. The racecars are prominent objects which fill up a good portion of the screen, and their color stands out from the background.

The subject of the second video is a skiier descending down a slope. This video is filmed from afar, and the skiier is very small. The skiier is wearing dark clothing against a white slope, but because the skiier is so small and the foreground includes the snow kicked up in the skiier's wake, of similar color to the background snow, this video presents a more difficult challenge for background subtraction.

2 Theoretical Background

2.1 The Koopman operator

Consider a set of numerical observations evenly spaced in time, represented as column vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$. Let us make the assumption that the transition from one observation to the next is a constant linear function. It can then be represented by the matrix A, which satisfies

$$\mathbf{x}_{j+1} = A\mathbf{x}_j \tag{1}$$

for all applicable j. This treats the state space like a solution to an unknown linear ODE, where transitions between states involve linear operations on the states.

2.2 Dynamic Mode Decomposition

Dynamic Mode Decomposition is a method of approximating a Koopman operator for our data. Suppose our numerical observations form the columns of a data matrix X. Furthermore, define the matrices X_1, X_2 by deleting the last and first column of X respectively.

$$X_1 := \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & ... & \mathbf{x}_{n-1} \end{bmatrix} \tag{2}$$

$$X_2 := \begin{bmatrix} \mathbf{x}_2 & \mathbf{x}_3 & ... & \mathbf{x}_n \end{bmatrix} \tag{3}$$

These matrices are almost related through left-multiplication by A, as is leveraged in Equation 4, where $\mathbf{r}$ is the error of the last column, and $e_{n-1}^\top$ is the standard basis vector to put these error vector in the right place for matrix addition.

$$X_2 = AX_1 + \mathbf{r}e_{n-1}^\top \tag{4}$$

We can't solve for A in closed form, but we can compute a matrix similar to it and indirectly obtain A's eigenvalues and eigenvectors, telling us enough we need to know about A. Compute the reduced Singular Value Decomposition of $X_1 = U\Sigma V^*$ where Σ is square and V^* only contains the row-space right singular vectors, and substitute this into Equation 4. We want to choose A such that the matrix X_2 is now expressed as a linear combination only of the columns of U, and that the residual vector is orthogonal to U^*. Under this assumption, we can left-multiply Equation 4 by U^*, right-multiply by V and simplify.

$$\begin{aligned} X_2 &= AU\Sigma V^* + \mathbf{r}e_{n-1}^\top \\ U^*X_2 &= U^*AU\Sigma V^* \\ \tilde{S} := U^*X_2V\Sigma^{-1} &= U^*AU \end{aligned} \tag{5}$$

The algebra expressed in Equation Set 5 provides a matrix $\tilde{S}$ similar to A. An eigenvalue/eigenvector pair of $\tilde{S} = U^*AU$, represented as $\tilde{S}\mathbf{v} = \lambda\mathbf{v}$, will also satisfy Equation Set 6, so then $U\mathbf{v}$ is an eigenvector of A with eigenvalue λ.

$$\begin{aligned} U^*AU\mathbf{v} &= \lambda\mathbf{v} \\ AU\mathbf{v} &= U\lambda\mathbf{v} \\ A(U\mathbf{v}) &= \lambda(U\mathbf{v}) \end{aligned} \tag{6}$$

We can store the eigenvectors of A as a matrix product $\Phi = \begin{bmatrix} U \begin{bmatrix} \mathbf{y}_1 & \mathbf{y}_2 & ... & \mathbf{y}_K \end{bmatrix} \end{bmatrix}$ where K is the rank of A, and equivalently the number of eigenvectors outside the nullspace.

For each eigenvalue μ_k, let $\omega_k = \log(\mu_k)/\Delta t$. Then our DMD approximation for the columns of X in terms of $\mathbf{x}_1$ can be represented as a linear combination of exponentials in Equation 7

$$\mathbf{x}_{DMD}(t) = \sum_{k=1}^{K} b_k \Phi_k e^{\omega_k t} \tag{7}$$

The coefficients $\mathbf{b}$ are constant, and when evaluating $\mathbf{x}_{DMD}(0) = \sum_{k=1}^{K} b_k\Phi_k = \Phi\mathbf{b}$ (where Φ_k is the k-th column of Φ), I can solve for $\mathbf{b}$ in terms of $\mathbf{x}_1$ and the pseudo-inverse of Φ.

2.3 Background subtraction

Each column of $\mathbf{x}_i$ can now be represented as a linear combination of the K DMD modes, which are themselves exponential functions. An exponential function $e^{\omega_k t}$ can represented exponential growth, decay, oscillation, or a constant depending on the value of ω_k. A constant-valued exponential function $e^{\omega_k t}$ would only come from an ω_k equal to or close to zero, which itself comes from an eigenvalue μ_k equal to or close to 1. Such values characterize a component of an image which is constant throughout the matrix; such components can be effectively regarded as "backgrounds", capturing the portion of a video which is primarily still.

3 Algorithm Implementation and Development

All data should be evenly-spaced in time. The given video data has many sequential duplicate frames, and these were stripped before forming the data matrix X.

Proper choice of threshold will depend on how close or how far the logarithmized eigenvalues ω are to 0. A threshold of $\epsilon = 1 \times 10^{-1}$ was found to be effective for both data samples in this paper. See Figure 2 and Figure 4.

Algorithm 1: Dynamic Mode Decomposition

Input: Data matrix X, evenly-spaced time evaluations $\mathbf{t}$ starting from $t = 0$
Set X_1, X_2 by deleting the last or first column of X respectively.
Compute the reduced SVD $X_1 = U\Sigma V^\top$
Compute $\tilde{S} = U^* X_2 V \Sigma^{-1}$
Compute the diagonalization $\tilde{S} = QDQ^{-1}$
Logarithmize the eigenvalues and store in $\omega = \log(\text{diag}(D)) \cdot \dfrac{1}{\Delta t}$
Compute the eigenvectors of A and store in $\Phi = UQ$
Solve $\Phi b = \mathbf{x}_1$ using the pseudo-inverse of Φ
Initialize an empty $K \times n$ matrix M for the modes
for all t_i **do**
  Store the point-wise product of $\mathbf{b}, \omega$ as the i-th row in M
end for
return M, ω, Φ

Algorithm 2: Background subtraction

Input: Data matrix X, M, ω, Φ from Alg. **??**, threshold $\epsilon > 0$
Initialize the background matrix B as an empty $l \times n - 1$ matrix where $X \in \mathbb{R}^{l \times n}$
for all ω_k with modulus less than ϵ **do**
  Compute the product P of the j-th column of Φ and the j-th row of M
  Update B to $B + P$
end for
Compute $F = X - |B|$ where $|\cdot|$ denotes the modulus of each element
return the foreground/background matrices F, B

4 Computational Results

For each video, the eigenvalues of A, μ, are mostly close to the unit circle. This suggests that each transition according to Equation 1 does not change the overall spectral power of an image very much. The only eigenvalues used to construct the background image are the ones that are closest to one, identified by the ones whose logarithm is closest to zero. There is one such mode in the racing video, and there are two such modes in the skiing video, though they appear as only one mode in Figure 4

5 Summary and Conclusions

Dynamic Mode Decomposition is successful at isolating the background in both the skiing and the racecar videos. Subtracting this background image from the original video yields the foreground well in both the racing video and the skiing video.

The skiing video is intended to be the "stress test" for background subtraction, and while some features of the background are retained in the foreground video, the true foreground is at least included in the right places. In Figure 3, the location of the skiier is circled in red in both the original and the foreground video. The image in the third row contains a large snow plume, which is correctly identified in the foreground.

In conclusion, the "foreground video" retains some features of the background which make the result imperfect for an application like CGI where visual reconstruction needs to be perfect. However, this could be used as one step in a sequence of algorithms to isolate the foreground with more photo processing techniques, or in a computer vision sequence of operations where minor visual noise can be thresholded.

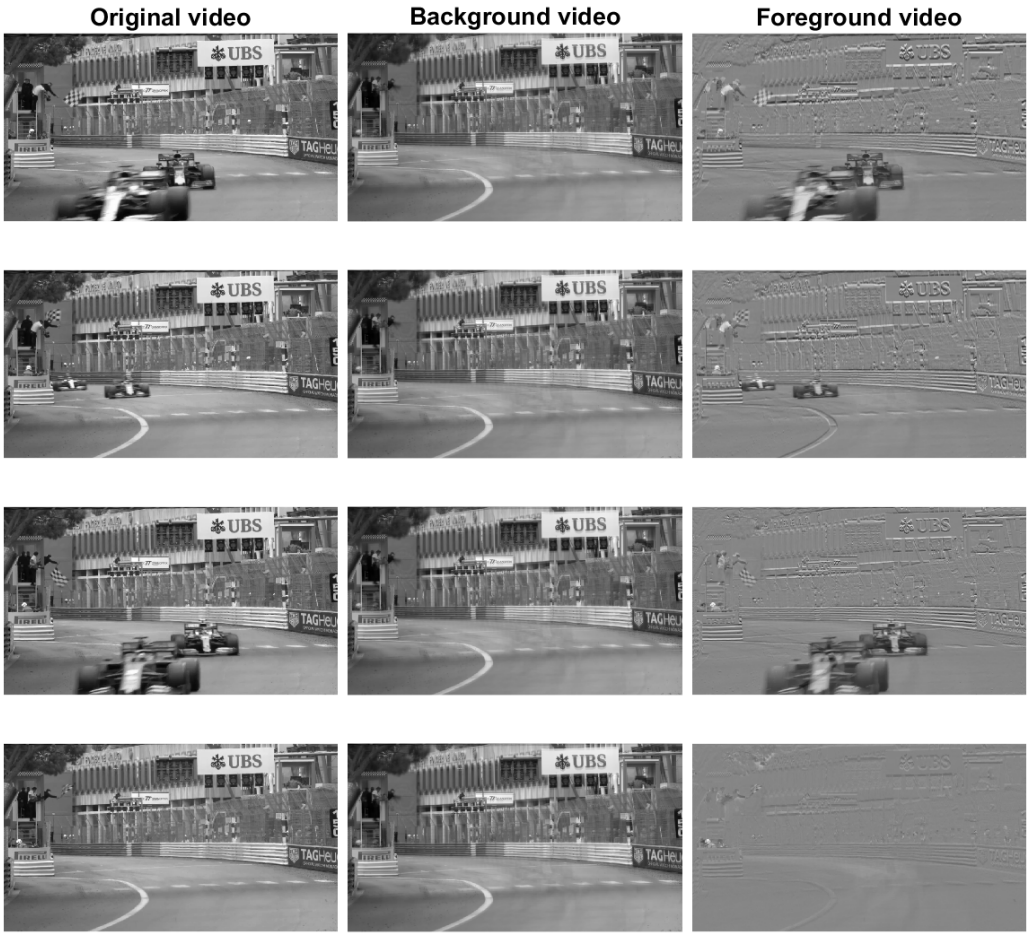

Figure 1: Four frames from the racing video

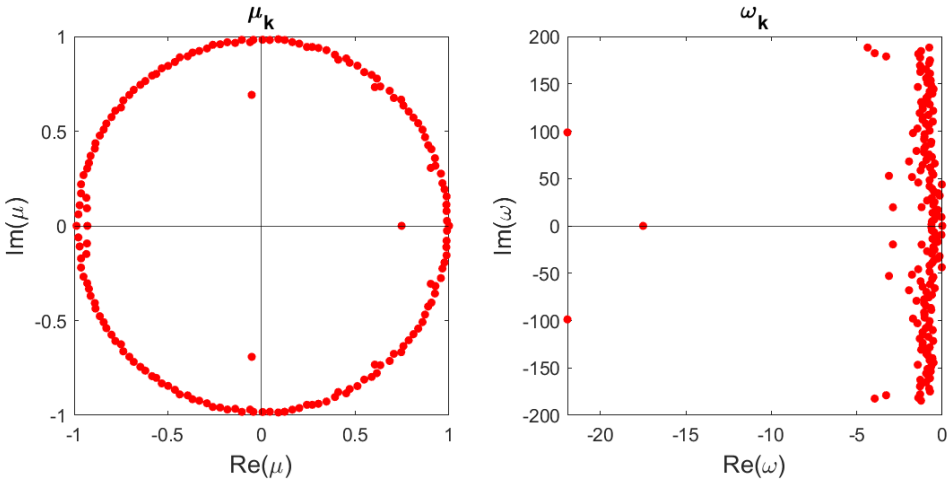

Figure 2: Distribution of eigenvalues and their logarithm for racing video

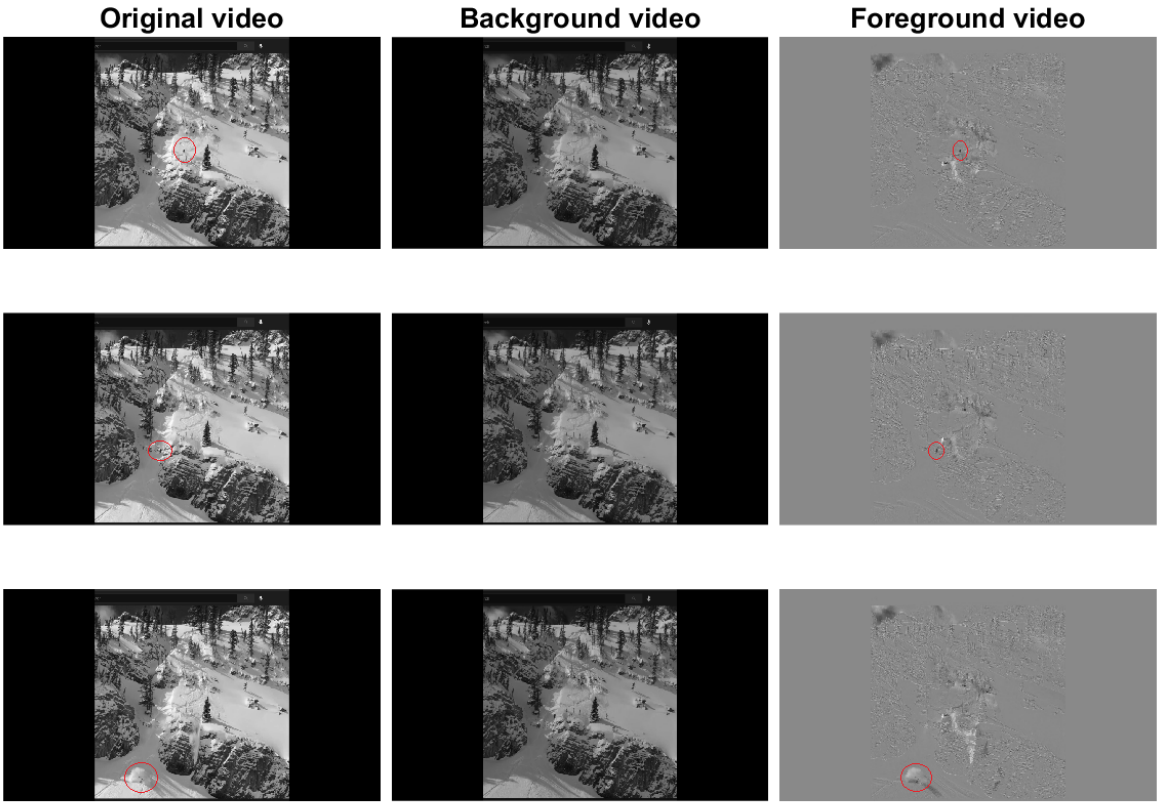

Figure 3: Three frames from the skiing video (location circled)

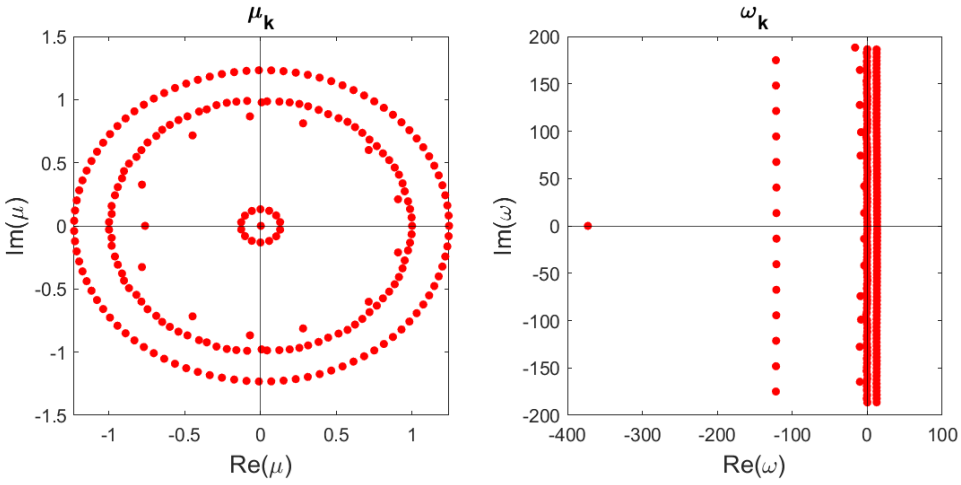

Figure 4: Distribution of eigenvalues and their logarithm for skiing video

Appendix A MATLAB Functions

- `[U,S,V] = svd(X, 'econ')` gives the reduced SVD of X.
- `A  b`, where system A is underdetermined, solves the pseudo-inverse.
- `subplot_tight()` by Nikolay S., Retrieved from MATLAB File Exchange. Allows for easy re-spacing of subplots in figures 2, 1.

Appendix B MATLAB Code

B.1 main.m

```
%% load data
clear all;

monte_carlo_low = 'data/monte_carlo_low.mp4';
monte_carlo_high = 'data/monte_carlo.mov';
ski_drop_low = 'data/ski_drop_low.mp4';

% params
choice = ski_drop_low;
portion = 1;

v = VideoReader(choice);
n = portion*v.NumFrames;

n = floor(n);
frames = read(v, [1 n]);
dim_h = size(frames, 1);
dim_w = size(frames, 2);
Xdup = zeros(dim_h*dim_w, n);

% convert rgb to grayscale
for j = 1:n
    frame = rgb2gray(frames(:,:,:,j));
    Xdup(:,j) = double(reshape(frame, [dim_h*dim_w 1]));
end
clear frames frame;

% remove duplicate frames
k = 1;
X = [];
maxdiffs = [];
for j = 1:n-1
    maxdiff = max(abs(Xdup(:,j)-Xdup(:,j+1)), [], 'all') ;
    maxdiffs(1,j) = maxdiff;
    if maxdiff > 15
        X(:,k) = Xdup(:,j);
        k = k + 1;
    end
end
n = k-1;
clear k maxdiff Xdup
```

```
figure(1);
stem(1:length(maxdiffs), maxdiffs, 'x');

X1 = X(:,1:end-1);
X2 = X(:,2:end);
dt = 1/v.FrameRate;

% X has columns of data [x1 x2 x3 ... xn] representing states
% I hypothesize that we can use a linear state transformation, a Koopman
% operator A such that X ~= [x1 Ax1 AAx1 ... A^(n-1)x1]
% We form S from X1,X2, and note that the same operations, when applied
%    to A, result in S = U'*A*U, so these matrices are similar.
% If y is an eigenvector of S, then Uy is an eigenvector of A.
% Thus we can get the eigenvectors of A, or the DMD modes, and their
% eigenvalues.
[U,Sigma,V] = svd(X1, 'econ');
S = U'*X2*V*diag(1./diag(Sigma));

clear X2 Sigma V
[evecs, evals] = eig(S);
mu = diag(evals);  % the eigenvalues
omega = log(mu)/dt;
phi = U*evecs; % the eigenvectors of A

% Now each x(t) is representated in a basis of phi_k * exp(omega_k*t)
% What are the linear combination coefficients? Notice that at t=0, we have
% the first state x1 and the exponential cancels. Thus x1 = phi_k*b.
% Let's use the pseudoinverse!
b = phi \ X1(:,1);
disp("Calculated b");
clear U S X1

%% Did we do well with the DMD deconstruction?
figure(2);
subplot(1,2,1);
plot(real(mu), imag(mu), 'r.', 'Markersize', 15);
xlabel('Re(\mu)', 'FontSize', 13);
ylabel('Im(\mu)', 'FontSize', 13);
xline(0, 'k');
yline(0, 'k');
title("\mu_k", 'FontSize', 13);

subplot(1,2,2);
plot(real(omega), imag(omega), 'r.', 'Markersize', 15);
xlabel('Re(\omega)', 'FontSize', 13);
ylabel('Im(\omega)', 'FontSize', 13);
xline(0, 'k');
yline(0, 'k');
title("\omega_k", 'FontSize', 13);

set(gcf, 'Position', [600 500 800 350]);
saveas(gcf, 'fig_2.png');
%% Now let's build our DMD reconstructions.
modes = zeros(length(b), n);
```

```
for iter = 1:n
    t = (iter-1)*dt;
    modes(:,iter) = b .* exp(omega*t);
end

inds = find(abs(omega) < 1e-1);
background = zeros(dim_h*dim_w,n);
for i = 1:length(inds)
    j = inds(i);
    background = background + phi(:,j)*modes(j,:);
end
background = abs(background);

foreground = X(:,1:end-1) - abs(background(:,end-1));
R = foreground .* (foreground < 0);

foreground = foreground - 0*R;
background = background(:,1:n-1) + 0*R;

disp("Launching movie player...");


vid_orig = reshape(mat2gray(X(:,1:n-1)), [dim_h dim_w n-1]);
vid_back = reshape(mat2gray(background), [dim_h dim_w n-1]);
vid_fore = reshape(mat2gray(foreground), [dim_h dim_w n-1]);

% delete(findall(0));
% implay(vid_orig);
% implay(vid_back);
% implay(vid_fore);
%%
inds = floor(linspace(1,n,6));
inds = inds(2:end-1);
figure(3);

for j = 1
    ind = inds(j+1);

    subplot_tight(3,3,j*3-2, [0.01]);
    imshow(vid_orig(:,:,ind));
    title("Original video", "FontSize", 13);

    subplot_tight(3,3,j*3-1, [0.01]);
    imshow(vid_back(:,:,ind));
    title("Background video", "FontSize", 13);

    subplot_tight(3,3,j*3, [0.01]);
    imshow(vid_fore(:,:,ind));
    title("Foreground video", "FontSize", 13);
end

for j = 2:3
    ind = inds(j+1);
```

```
    subplot_tight(3,3,j*3-2, [0.01]);
    imshow(vid_orig(:,:,ind));

    subplot_tight(3,3,j*3-1, [0.01]);
    imshow(vid_back(:,:,ind));

    subplot_tight(3,3,j*3, [0.01]);
    imshow(vid_fore(:,:,ind));
end

set(gcf, 'Position', [2203 168 744 683]);
saveas(gcf, 'fig_3.png');

% background is the DMD mode that does not change in time...
% how'd you pick the rank?
% Why are we interested in the omegas close to zero? Because e^0=1 and that
% corresponds to the mode which does not change in time
```

B.2 subplot_tight.m, by Nikolay S.

```
function vargout=subplot_tight(m, n, p, margins, varargin)
%% subplot_tight
% A subplot function substitude with margins user tunabble parameter.
%
%% Syntax
%  h=subplot_tight(m, n, p);
%  h=subplot_tight(m, n, p, margins);
%  h=subplot_tight(m, n, p, margins, subplotArgs...);
%
%% Description
% Our goal is to grant the user the ability to define the margins between neighbouring
%  subplots. Unfotrtunately Matlab subplot function lacks this functionality, and the
%  margins between subplots can reach 40% of figure area, which is pretty lavish. While at
%  the begining the function was implememnted as wrapper function for Matlab function
%  subplot, it was modified due to axes del;etion resulting from what Matlab subplot
%  detected as overlapping. Therefore, the current implmenetation makes no use of Matlab
%  subplot function, using axes instead. This can be problematic, as axis and subplot
%  parameters are quie different. Set isWrapper to "True" to return to wrapper mode, which
%  fully supports subplot format.
%
%% Input arguments (defaults exist):
%   margins- two elements vector [vertical,horizontal] defining the margins between
%        neighbouring axes. Default value is 0.04
%
%% Output arguments
%   same as subplot- none, or axes handle according to function call.
%
%% Issues & Comments
%  - Note that if additional elements are used in order to be passed to subplot, margins
%     parameter must be defined. For default margins value use empty element- [].
%  -
%
%% Example
% close all;
```

```
% img=imread('peppers.png');
% figSubplotH=figure('Name', 'subplot');
% figSubplotTightH=figure('Name', 'subplot_tight');
% nElems=17;
% subplotRows=ceil(sqrt(nElems)-1);
% subplotRows=max(1, subplotRows);
% subplotCols=ceil(nElems/subplotRows);
% for iElem=1:nElems
%    figure(figSubplotH);
%    subplot(subplotRows, subplotCols, iElem);
%    imshow(img);
%    figure(figSubplotTightH);
%    subplot_tight(subplotRows, subplotCols, iElem, [0.0001]);
%    imshow(img);
% end
%
%% See also
%  - subplot
%
%% Revision history
% First version: Nikolay S. 2011-03-29.
% Last update:   Nikolay S. 2012-05-24.
%
% *List of Changes:*
% 2012-05-24
%  Non wrapping mode (based on axes command) added, to deal with an issue of disappearing
%     subplots occuring with massive axes.

%% Default params
isWrapper=false;
if (nargin<4) || isempty(margins)
    margins=[0.04,0.04]; % default margins value- 4% of figure
end
if length(margins)==1
    margins(2)=margins;
end

%note n and m are switched as Matlab indexing is column-wise, while subplot indexing is row-wise :(
[subplot_col,subplot_row]=ind2sub([n,m],p);


height=(1-(m+1)*margins(1))/m; % single subplot height
width=(1-(n+1)*margins(2))/n;  % single subplot width

% note subplot suppors vector p inputs- so a merged subplot of higher dimentions will be created
subplot_cols=1+max(subplot_col)-min(subplot_col); % number of column elements in merged subplot
subplot_rows=1+max(subplot_row)-min(subplot_row); % number of row elements in merged subplot

merged_height=subplot_rows*( height+margins(1) )- margins(1);   % merged subplot height
merged_width= subplot_cols*( width +margins(2) )- margins(2);   % merged subplot width

merged_bottom=(m-max(subplot_row))*(height+margins(1)) +margins(1); % merged subplot bottom position
merged_left=min(subplot_col)*(width+margins(2))-width;              % merged subplot left position
pos=[merged_left, merged_bottom, merged_width, merged_height];
```

```
if isWrapper
   h=subplot(m, n, p, varargin{:}, 'Units', 'Normalized', 'Position', pos);
else
   h=axes('Position', pos, varargin{:});
end

if nargout==1
   vargout=h;
end
```