Dynamic Mode Decomposition

Christine M. Baker, GitHub: `cmbaker94`

March 17, 2021

Abstract

Dynamic Mode Decomposition (DMD) is applied to a video of a skier and race car track to extract the background of the image from the moving objects. The foreground and background are extracted based on assumptions of the DMD's approximate low-rank reconstruction. The resulting background and foreground are demonstrated.

1 Introduction and Overview

Dynamic Mode Decomposition (DMD) method is used on video clips that contain a foreground and background option. These are separated for a video stream. By assuming that the background video is in the DMD's approximate low-rank reconstruction, this can be subtracted from the original frames to identify the DMD's approximate sparse reconstruction. The resulting foreground and background objects are presented.

The theoretical background on DMD are provided in Section 2. The algorithm implementation and development to resolve the principle components are presented in Section 3. The computational results for the two videos are shown in Section 4. A conclusions are summarized in Section 5. Details about the MATLAB functions and the function written for this project are in Appendix A and the MATLAB code to run the analysis is in Appendix B.

2 Theoretical Background

Dynamic Mode Decomposition (DMD) can be used when validation of model equations is not possible or the governing equations are not known. DMD is a data-based algorithm that uses the low-dimensionality of experimental data and does not require governing equations, but instead relies on snapshots of measurements to predict and control the governing equations and dimensionality (Kutz, 2013). The DMD method decomposes experimental data into a set of dynamic modes based on snapshots of the data in time.

DMD requires that there is N spatial points saved per time snapshot, M number of snapshots taken, and regularly spaced intervals of the data collection in time (*i.e.*, $t_{m+1} = t_m + \Delta t$, where t_1 is the start, t_m is the end, and Δt is the collection rate). The data is reshaped to an $N \times M$ matrix with a constant time interval:

$$\mathbf{X}_j^k = [U(\mathbf{x}, t_1)U(\mathbf{x}, t_2)...U(\mathbf{x}, tM)] \tag{1}$$

The Koopman operator is a linear, infinite-dimensional operator that represented nonlinear, infinite-dimensional dynamics without linearization. These operator modes are approximated by the DMD method, such that the eigenvalues and eigenvectors from low-dimensional modes of a linear model can approximate the dynamics, even if the dynamics is nonlinear. The Koopman operator $\mathbf{A}$, a linear time-indepdent operator, is defined such that

$$\mathbf{x}_{j+1} = \mathbf{A}\mathbf{x}_j \tag{2}$$

where j indicates the time, $\mathbf{A}$ is the nonlinear operator, and $\mathbf{x}_j$ is an N-dimensional vector of the data at j times. A matrix $\mathbf{X}_1^{M-1}$ is constructed to best represent the data collected as:

$$\mathbf{X}_1^{M-1} = [\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3...\mathbf{x}_{M-1}] = [\mathbf{x}_1\mathbf{A}\mathbf{x}_1\mathbf{A}^2\mathbf{x}_2...\mathbf{A}^{M-2}\mathbf{x}_1] \tag{3}$$

The columns of this matrix are elements in a Krylov space and the matrix fits the first $M-1$ data collection using the operator matrix $\mathbf{A}$. The dimensionality reduction method takes advantage of the low-dimensionality of the data and uses singular valued decomposition (SVD):

$$\mathbf{X}_1^{M-1} = \mathbf{U}\Sigma\mathbf{V}^* \tag{4}$$

Teh DMD will fail if the data is not low rank, whereas if the data is low rank it will use the low-dimensional structure to project into the future state of the system. The matrix can be generalized to:

$$\mathbf{A}\mathbf{X}_1^{M-1} = \mathbf{X}_2^M = \mathbf{X}_1^{M-1}\mathbf{S} + re_{M-1}^* \tag{5}$$

where e_{M-1} is a $M-1$th unit vector, $\mathbf{S}$ is the approximate some of the eigenvalues of $\mathbf{A}$. The low-rank matrix can be computed as:

$$\tilde{\mathbf{S}} = \mathbf{U}^*\mathbf{X}_2^M\mathbf{V}\Sigma^{-1} \tag{6}$$

Furthermore, using the low rank approximation for the eigenvalues and eigenvectors, the approximate solution for all future times $\mathbf{X}_{DMD}(t)$ is given as:

$$\mathbf{X}_{DMD}(t) = \sum_{k=1}^{K} b_k(0)\psi_k(\mathbf{x}exp(\omega t)) = \Psi diag(exp(\omega t)\mathbf{b}) \tag{7}$$

where $\psi_k = \mathbf{U}\mathbf{y}_k$, $b_k(0)$ is the initial amplitude at each mode, Ψ is the matrix with columns that are eigenvectors, $diag(\omega t)$ is the diagonal matrix. with the eigenvalues $exp(\omega_k t)$, and $\mathbf{b}$ is a vector of coefficients b_k. The solution for $\mathbf{b}$ can be defined as:

$$\mathbf{b} = \Psi^+\mathbf{x}_1 \tag{8}$$

and can be found using a pseudo-inverse, where Ψ^+ is the Moore-Penrose pseudo-inverse. The DMD does not requite an equation and is still able to predict the future states.

To find the sparse reconstruction X_{sparse}^{DMD}, the original images X are subtracted by the low-rank reconstruction $X_{low-rank}^{DMD}$:

$$X = X_{sparse}^{DMD} + X_{low-rank}^{DMD} \tag{9}$$

The negative residuals, R from the difference between $X - X_{low-rank}^{DMD}$ are removed as presented in the homework assignment.

3 Algorithm Implementation and Development

The following processes is applied on each video. The videos are read into MATLAB, converted to black and white, and trimmed to remove any blank space (for the skier video). Example snapshots from the monte carlo and ski drop video are shown in Figure 1.

The sample data at N prescribed locations M times is reshaped into vectors with rows that are the reshaped pixels and the columns represent each snapshot in matrix X. The data snapshots are assumed to be evenly spaced in time by a fixed Δt. The singular valued decomposition (SVD) is computed for the matrix and the diagonals of σ are used to compute the covariance. The rank of the videos was selected by the first mode that is below 1% of the covariance percent.

Then from the data matrix X, the sub-matrices X_{m-1} and X_m. The SVD decomposition of X_{m-1} is computed and the matrix $\tilde{S}$ is computed and the eigenvalues and eigenvectors are computed. Then the initial state of the system is projected onto the Dynamic Mode Decomposition (DMD) modes using the pseudo-inverse. The solution is computed at any future time using the DMD modes along the projection to the initial conditions and the time dynamics computed using the eigenvalues of $\tilde{S}$.

The initial frames is subtracted by the low-rank reconstruction to obtain the sparse reconstruction, and the residuals, R are removed.

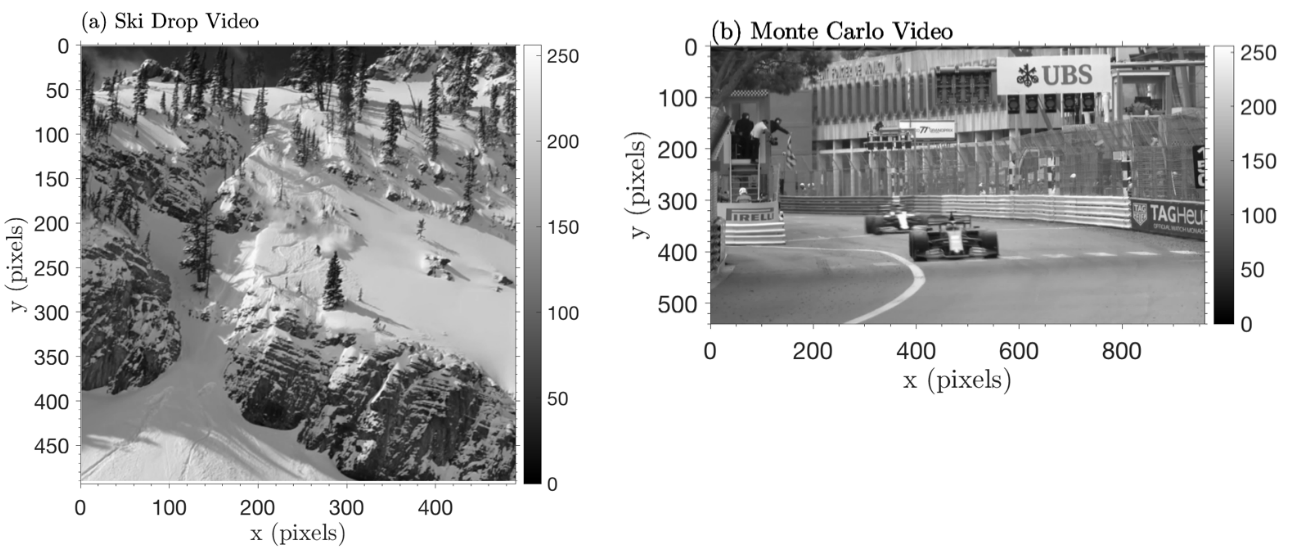

Figure 1: Example snapshots from the (a) trimmed ski drop video (frame 110) and (b) monte carlo (frame 180).

4 Computational Results

The results from the DMD low-rank (foreground) and sparse (foreground) reconstruction for the ski drop video (example snapshot in Figure 3) and monte carlo (example snapshot in Figure 4. The rank of the ski drop is 1 and of the monte carlo is 2, which as selected as described in the previous section. The ski drop video kept the snow and trees in the background, and the skier as well as the snow moving around the skier was the foreground. The monte carlo video background was the race car track and road, while the foreground was the race cars zooming by. There was some shake in the camera, and therefore, the foreground image does have some outline of the background features.

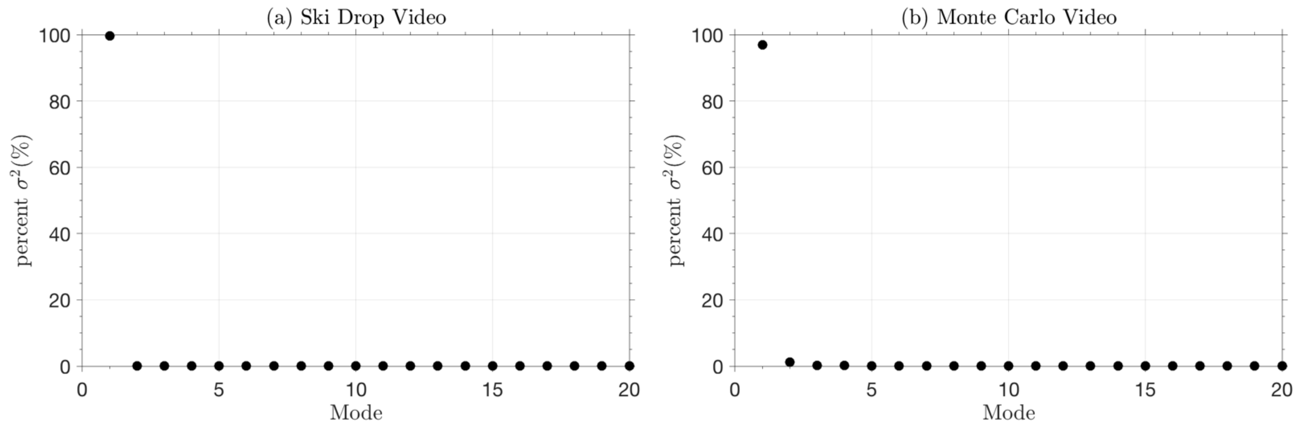

Figure 2: The percent of the total variance at the first 20 modes of the singular valued decomposition from the (a) ski drop and (b) monte carlo video.

5 Summary and Conclusions

Dynamic mode decomposition was used to sparse the foreground and the background from a video of a skier and of race cars. DMD is able to resolve the dynamics of an unknown governing equation and can predict the future timesteps. In this assignment, DMD successfully separated the skier from the mountain background and the race cars from the race track.

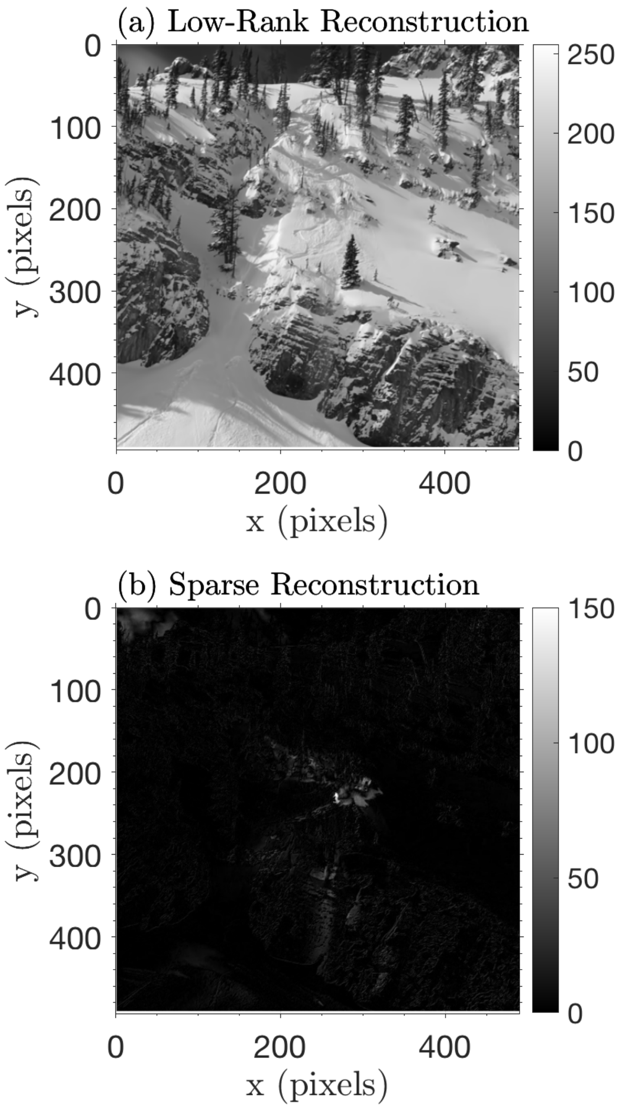

Figure 3: The (a) background (low-rank reconstruction) of the image and the (b) foreground (sparse reconstruction) from frame 110 of the ski drop video.

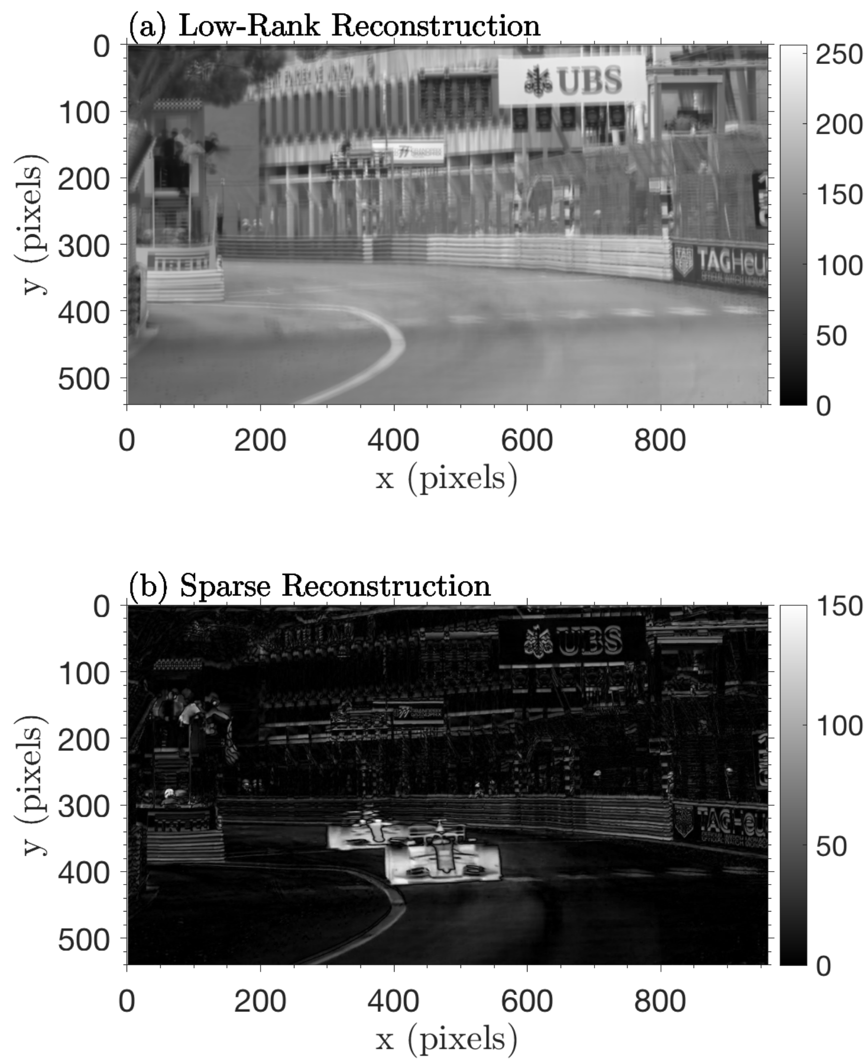

Figure 4: The (a) background (low-rank reconstruction) of the image and the (b) foreground (sparse reconstruction) from frame 180 of the monte carlo video.

Appendix

A MATLAB functions used and brief implementation explanation

- `axis`: indicate the limits for the current plotted axes
- `abs`: compute absolute value
- `calc_dmd`: function to compute the DMD
- `figure`: open new figure
- `find`: find values meeting some threshold
- `diag`: extract diagonals
- `double`: change value to type double
- `eig`: compute eigenvalues
- `grid`: add grid lines to a figure
- `imshow`: plot image
- `*label`: label the x, y, z axes of plots
- `length`: find the length of the largest array dimension (grab the time-dimension in loops)
- `load`: load data from a .m file into workspace (load subdata.m)
- `log`: compute log
- `pcolor`: method to plot surfaces in 2d
- `prep_vid`: read video and trim
- `print`: export and save figure
- `repmat`: created matrix with repeated values
- `reshape`: reshape data
- `read`: read the video frames
- `rgb2gray`: convert rgb image to gray
- `scatter`: create scatter plot
- `set`: manipulate a figure
- `size`: grab size of a matrix

- `svd`: compute singular value decomposition values
- `text`: define text for a figure
- `VideoReader`: read video
- `zeros`: create a matrix of zeros

Function to compute DMD:

```
function [u_dmd] = calc_dmd(f,r)
% Compute the dmd
% input: f - data matrix, r - rank
% output: u_dmd = dmd'ed u

[nim, resxy] = size(f);
t=linspace(0,nim,nim); dt=t(2)-t(1);
% x = linspace(0,resxy,resxy);

% figure(2)
% subplot(4,3,1), plot(diag(s)/sum(diag(s)),'ko','Linewidth',[2])
% subplot(4,1,2), plot(t,v(:,1)/max(v(:,1)),t,v(:,2)/max(v(:,2)),'
    Linewidth',[2])
% subplot(4,1,3), plot(x,u(:,1)/max(u(:,1)),'Linewidth',[2])
% subplot(4,1,4), plot(x,u(:,2)/max(u(:,2)),'Linewidth',[2])

X = f.'; X1 = X(:,1:end-1); X2 = X(:,2:end);
[U2,Sigma2,V2] = svd(X1,'econ'); U=U2(:,1:r); Sigma=Sigma2(1:r,1:r);
    V=V2(:,1:r);

% DMD J-Tu decomposition:  Use this one

Atilde = U'*X2*V/Sigma;
[W,D] = eig(Atilde);
Phi = X2*V/Sigma*W;

mu = diag(D);
omega = log(mu)/dt;

u0=f(1,:).';
y0 = Phi\u0;  % pseudo-inverse initial conditions
u_modes = zeros(r,length(t));
for iter = 1:length(t)
     u_modes(:,iter) =(y0.*exp(omega*t(iter)));
end
u_dmd = Phi*u_modes;
```

Function to read video:

```matlab
function [data,framerate] = prep_video(file,trim)
% read and reformat video

v = VideoReader(file);
framerate = v.FrameRate;
ski_rgb = read(v);

for i = 1:size(ski_rgb,4)
    data(:,:,i) = rgb2gray(ski_rgb(trim(3):trim(4),trim(1):trim(2),:,
       i));
end
```

B MATLAB code

Code available at: `https://github.com/cmbaker94/Baker_AMATH582`

```matlab
clear all
close all
clc

addpath(genpath('/Users/cmbaker9/Documents/MTOOLS'))

%% STEP 0: Locate and load data

datapath    = '/Users/cmbaker9/Documents/UW_Classes/
   AMATH_Data_Analysis/HW5/data/';
figfolder   = '/Users/cmbaker9/Documents/UW_Classes/
   AMATH_Data_Analysis/HW5/figures/';

%% Load data

vid = 'mon';

if vid == 'ski'
    vname = 'ski_drop_low';
    trim = [234 726 44 533];
    rank = 1; % below 1% covariance (2nd is 0.1%)
elseif vid == 'mon'
    vname = 'monte_carlo_low';
    trim = [1 960 1 540];
    rank = 2; % below 1% covariance (3rd is just below 1%)
end

videofile = [datapath,vname,'.mp4'];
[frames, framerate] = prep_video(videofile,trim);
```

```matlab
%% run svd on data

[resx, resy] = size(frames,1:2);
nim = size(frames,3);

f = double(reshape(frames,resx*resy,nim))';

[u,s,v]=svd(f','econ');

% covariance
lambda=diag(s).^2;

figure('units','inches','position',[1 1 10 6],'Color','w');
mode = 1:length(lambda);
scatter(mode,lambda/sum(lambda)*100,100,'k','fill')
% plot(diag(s)/sum(diag(s)),'ko','Linewidth',[2])
hold on
% title('(a) Ski Drop Video','interpreter','latex','fontsize',20);
title('(b) Monte Carlo Video','interpreter','latex','fontsize',20,'Color',[0 0 0]);
xlim([0 20])
xlabel('Mode','interpreter','latex','fontsize',20)
ylabel('percent $\sigma^2 (\%)$','interpreter','latex','fontsize',20)
grid on
box on
h1=gca;
set(h1,'fontsize',20);
set(h1,'tickdir','out','xminortick','on','yminortick','on');
set(h1,'ticklength',1*get(h1,'ticklength'));
Sname1 = [figfolder,vid,'_covariance'];
print(Sname1,'-dpng')

[u_dmd] = calc_dmd(f,rank);

%% Seperate sections of image

Xlr = abs(reshape(u_dmd,resx,resy,nim));
Xsparse = double(frames)-abs(Xlr);
% Xsparse = abs(double(frames)-abs(Xlr));

R = Xsparse;
R(R<0) = 0;
Xsparse = Xsparse-R;
Xlr = abs(Xlr)+R;

%% Create Plot
```

```
figure('units','inches','position',[1 1 8 20],'Color','w');
% timeplot = 10;
for timeplot = 100:10:200
    subplot(2,1,1)
    pcolor(Xlr(:,:,timeplot)); shading interp; colorbar
    hold on
    colormap('gray')
    caxis([0 256])
    set(gca, 'YDir','reverse')
    text(0,-25,'(a) Low-Rank Reconstruction','interpreter','latex','fontsize',20,'Color',[0 0 0]);
    xlabel('x (pixels)','interpreter','latex','fontsize',20)
    ylabel('y (pixels)','interpreter','latex','fontsize',20)
    grid on
    box on
    axis equal
    xlim([0 resy])
    ylim([0 resx])
    h1=gca;
    set(h1,'fontsize',20);
    set(h1,'tickdir','out','xminortick','on','yminortick','on');
    set(h1,'ticklength',1*get(h1,'ticklength'));

    subplot(2,1,2)

    pcolor(Xsparse(:,:,timeplot)); shading interp; colorbar;
    hold on
    colormap(flipud('gray'))
    caxis([0 150])
    set(gca, 'YDir','reverse')
    text(0,-25,'(b) Sparse Reconstruction','interpreter','latex','fontsize',20,'Color',[0 0 0]);
    xlabel('x (pixels)','interpreter','latex','fontsize',20)
    ylabel('y (pixels)','interpreter','latex','fontsize',20)
    grid on
    box on
    axis equal
    xlim([0 resy])
    ylim([0 resx])
    h1=gca;
    set(h1,'fontsize',20);
    set(h1,'tickdir','out','xminortick','on','yminortick','on');
    set(h1,'ticklength',1*get(h1,'ticklength'));
    drawnow
    Sname = [figfolder,'/',vname,'_',num2str(timeplot)];
```

```
    print(Sname, '-dpng')
end

figure('units','inches','position',[1 1 8 8],'Color','w');
for timeplot = 100:10:200
    pcolor(double(frames(:,:,timeplot))); shading interp; colorbar
    hold on
    colormap('gray')
    caxis([0 256])
    set(gca, 'YDir','reverse')
    text(0,-25,'(b) Monte Carlo Video','interpreter','latex','
       fontsize',20,'Color',[0 0 0]);
    xlabel('x (pixels)','interpreter','latex','fontsize',20)
    ylabel('y (pixels)','interpreter','latex','fontsize',20)
    grid on
    box on
    axis equal
    h1=gca;
    set(h1,'fontsize',20);
    set(h1,'tickdir','out','xminortick','on','yminortick','on');
    set(h1,'ticklength',1*get(h1,'ticklength'));
    xlim([0 resy])
    ylim([0 resx])
    Sname = [figfolder,'/',vname,'_orig_',num2str(timeplot)];
    print(Sname, '-dpng')
end
```

References

J Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.