Homework 4: Background Subtraction in Video Streams

Eunji Lindsey Lee
(*e-mail:* elee0025@uw.edu)

Abstract

This paper explores the methods of Dynamic Mode Decomposition (DMD) and its interesting application in segregating background from foreground from videos. Background in the video is the part of video that is static overtime thus the DMD terms with low Fourier frequencies that's close to zero is considered as background, and the rest is considered as the foreground part of the video (DMD terms with low Fourier frequencies that's not close to zero). The paper attempts to analyze the result after performing DMD on the selected video, and evaluate if the produced videos of background and foreground seem valid.

Contents

1 Introduction and Overview

The paper uses Dynamic Mode Decomposition (DMD) and effectively separate a stationary background from a dynamic foreground from a chosen video clip, based on its snapshots of individual frames. DMD is a matrix decomposition which is based on the power of Singular Value Decomposition (SVD). The method has gained popularity due to an increasing need of data science in various fields like bio-engineering and physical sciences [**1**]. It implements a Fourier decomposition together with a reduction of spatial dimensionality, and computes DMD modes and eigenvalues from video frames, which represent a non-linear dynamic system. DMD modes and eigenvalues yields oscillatory

time components of the video frames that have contextual implications. DMD modes that's close to zero are the low-rank components of a data matrix (collection of pixels of each frames of video), and the rest of the DMD modes are the sparse components of a data matrix.

The major advantage of using DMD modes in computer visions is its fast speed of computational operations, which enable the real-time computing especially on appropriate videos, especially surveillance videos where the background is stationary [**1**]. The analysis done on the paper is not performing real-time analysis on videos but it discusses about the performance of DMD application on surveillance video and the limitations of applications as well.

2 Theoretical Background

2.1 Dynamic Mode Decomposition

The Dynamic Mode Decomposition (DMD) has originally been developed and used for fluid mechanics and nonlinear waves as a data-based method, as it analyzes the complex non-linear dynamic system without knowing its dominant equations of the system. Thus, DMD is also called "equation-free" method. DMD takes data that has "spatio-temporal" information and analyzes the times series of spatial data accordingly. Due to such a property, DMD allows data to be analyzed for recorded time periods so as to to guess past states as well as to even predict future states of the system. Given data collected for DMD method is evenly spaced in time, so that it will appropriately approximate the "low-dimensional modes of the linear, time-independent *Koopman operator*", where the *Koopman operator* represents the non-linear dynamics as an linear, infinite dimensional operator on the Hilbert space. In other words, DMD interprets the complex nonlinear system in a simpler, linear form of data. The main key to the DMD operation is the approximation of the eigenvalues of *Koopman operator*.

Video is simply a collection of frames in equally spaced time, which makes it appropriate for the analysis of DMD. Consider n as the data collected at a given time (pixels of a frame, in this case) and m as the total number of snapshots (total number of frames in a video, in this case). A data matrix can then be formulated by combining m columns vectors of size nx1, and then its subsets of two large data matrices X1 and X2 are used to minimize the approximation error. Perform Singular Value Decomposition (SVD) to truncate data and reduce dimensions, and perform Eigenvalue Decomposition to obtain DMD eigenvalues and eigenvectors, which are then converted into Fourier modes, which is expressed in terms of growth and decay rate as well as oscillation. Therefore, each frames are now decomposed into DMD modes.

The following decomposition makes the static portion of video to have an Fourier modes that's close to zero as it either moves very slowly in time or doesn't move. This point is represented as a origin or close to origin in the complex space. Most importantly, this key fact allows us to separate background (low-rank matrix) from foreground (sparse matrix) using DMD method.

3 Algorithm Implementation and Development

3.1 Pre-processing videos

We used python library *cv2* to read a mp4 video file as a collections of frames, where each frame is read as a matrix of pixels. For a better segregation of video into backgrounds and foregrounds, the appropriate video where background is fixed and only objects or persons are moving in the foreground, is strongly required. The video I chose for the analysis was from Youtube where it took a video of a train from a stationary camera, so that the background is stationary and the only moving object was a train [**3**]. For simplicity, we converted to gray-scale and converted the size of each frame of video to 250 by 150. In order to combine the whole frames of video as a matrix, we reshaped each frames as a column vector of size of 250*150 pixels by 1, then combined column vectors to form a matrix. If the video has m frames, the size of a matrix becomes 250*150 by m.

3.2 DMD decomposition

From 250*150 by m data matrix (collection of video frames), X, we created the DMD input-output matrices called $X1$ and $X2$. The matrix looks like the following:

$$X = [x_1 \quad x_2 \quad \dots \quad x_m] \tag{0}$$

where each x_i represents a column vector of an original data matrix, X. Then matrices $X1$ and $X2$ look like the following:

$$X1 = [x_1 \quad x_2 \quad \dots \quad x_{m-1}]$$

$$X2 = [x_2 \quad x_3 \quad \dots \quad x_m]$$

Then we performed "reduced" SVD decomposition of X1 and obtained matrices U, S, V'. Then we truncated zero values from SVD by calculating rank of a matrix X1, r. Then we kept 1 to r columns of U and V' and kept rows and columns of S from 1 to r. U matrix from the SVD decomposition represents the Proper Orthogonal Decomposition modes, where modes are ranked in terms of energy content.

Then we built a tilde matrix which is equivalent to $U * X2 * V * S^{-1}$. Then we did the eigenvalue decomposition of a tilde matrix to create eigenvalues μ and eigenvectors W. Now, we created a DMD modes matrix Φ, which is equivalent to $X2 * V * S^{-1} * W$. To calculate Fourier modes, which is related to eigenvalues of DMD modes (or called "*Koopman modes*"), we calculate $\omega = \frac{\log(\mu)}{dt}$, where dt means the time interval between frames. From DMD modes, we then captures background and foreground. Background is represented by the omega values that's close to zero value. The threshold we used to capture omega value was 1e-02, and we chose omega values that's less than the threshold value. In other words, given ω_p, where $p \in \{1, 2, ..., r\}$, p value that satisfies $\|\omega_p\| \approx 0$, and ω_j represents the omega values that corresponds to foreground. Therefore,

$$X_{DMD} = b_p \varphi_p e^{\omega_p t} + \sum_{j \neq p} b_j \varphi_j e^{\omega_j t}$$

The first part, $b_p\varphi_p e^{\omega_p t}$ represents the Background video, which we also called as Low-rank matrix($X_{DMD}^{LOW-RANK}$), and the second part, $\sum_{j\neq p} b_j\varphi_j e^{\omega_j t}$ represents the Foreground video, which is also called as Sparse matrix(X_{DMD}^{SPARSE}).

Since sparse matrix is in complex space, we performed simple modification to put the matrix in real space. Here is the modifications performed:

$$X_{DMD}^{SPARSE} = X - |X_{DMD}^{LOW-RANK}|$$

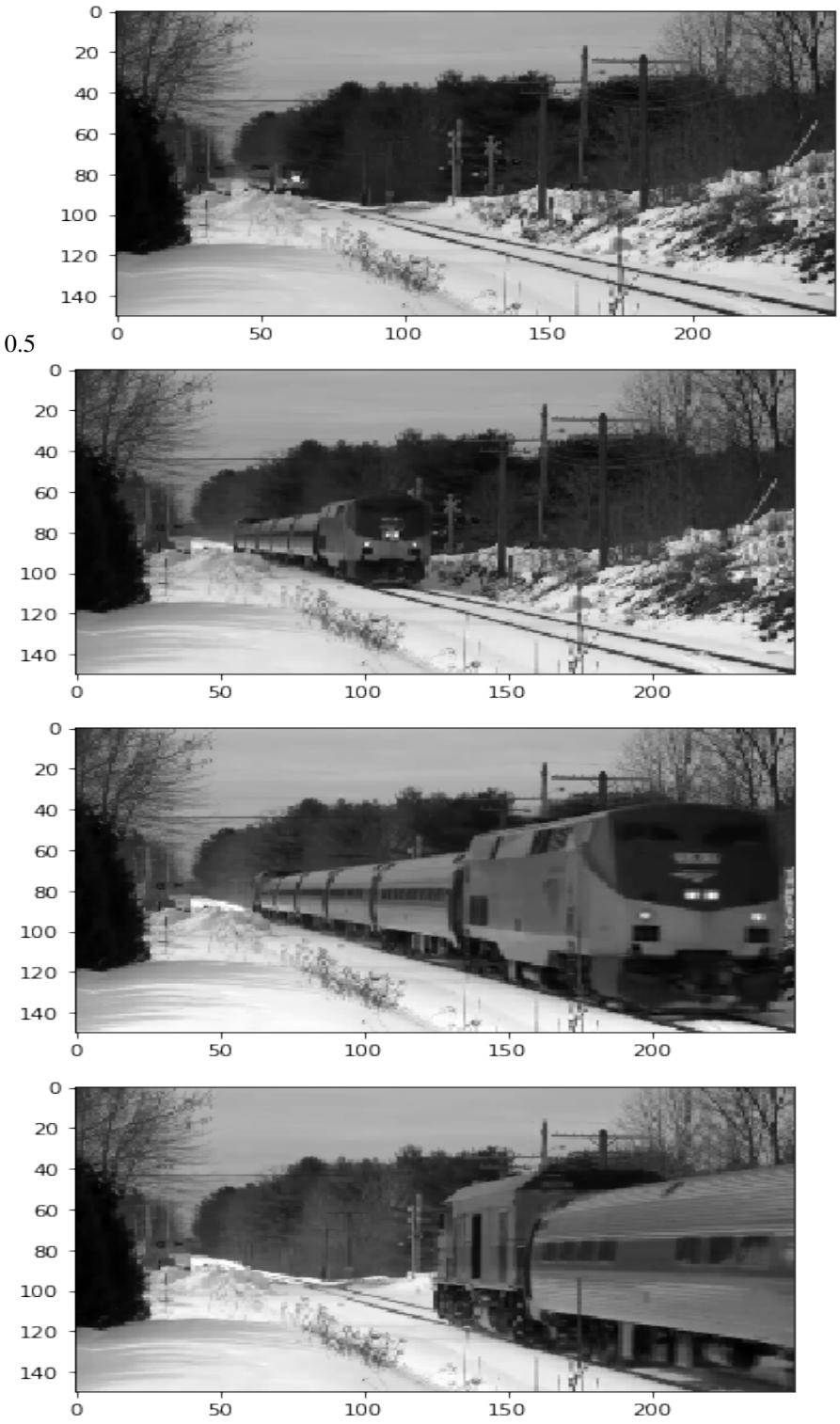
From sparse matrix, we simply captured residual negative values and put it into the matrix of the same shape as matrix R. Then,

$$X_{DMD}^{LOW-RANK} \leftarrow R + |X_{DMD}^{LOW-RANK}|$$

$$X_{DMD}^{SPARSE} \leftarrow X_{DMD}^{SPARSE} - R$$

4 Computational Results

The video we used for the analysis is only 25 seconds long, and it captures 29 frames per second. It means it has total of 25*29=725 frames. By performing reduced SVD, we computed the rank of the data matrix, and it returned $r = 719$. As mentioned in the previous section, in order to get omega values that's close to 0, we set the threshold value as 1e-02. There was only one omega value that was captured as a background (omega value less than the threshold).



0.5 0.5

0.5

0.5

Figure 1. The following pictures are the captures of the combined X-low-rank matrix and X-sparse matrix

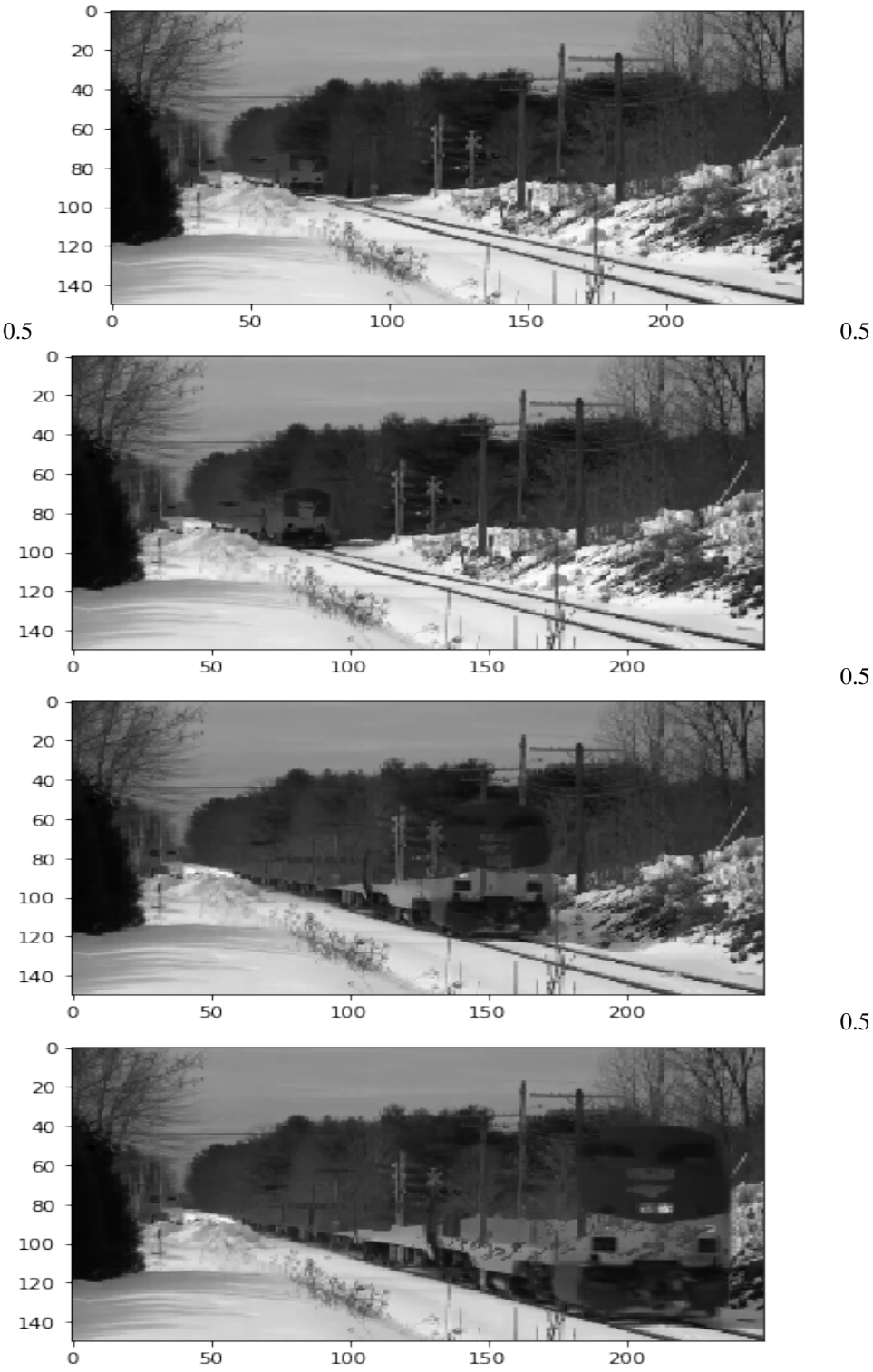


Figure 2. The following pictures are the captures of the X-low-rank matrix, which represents the background

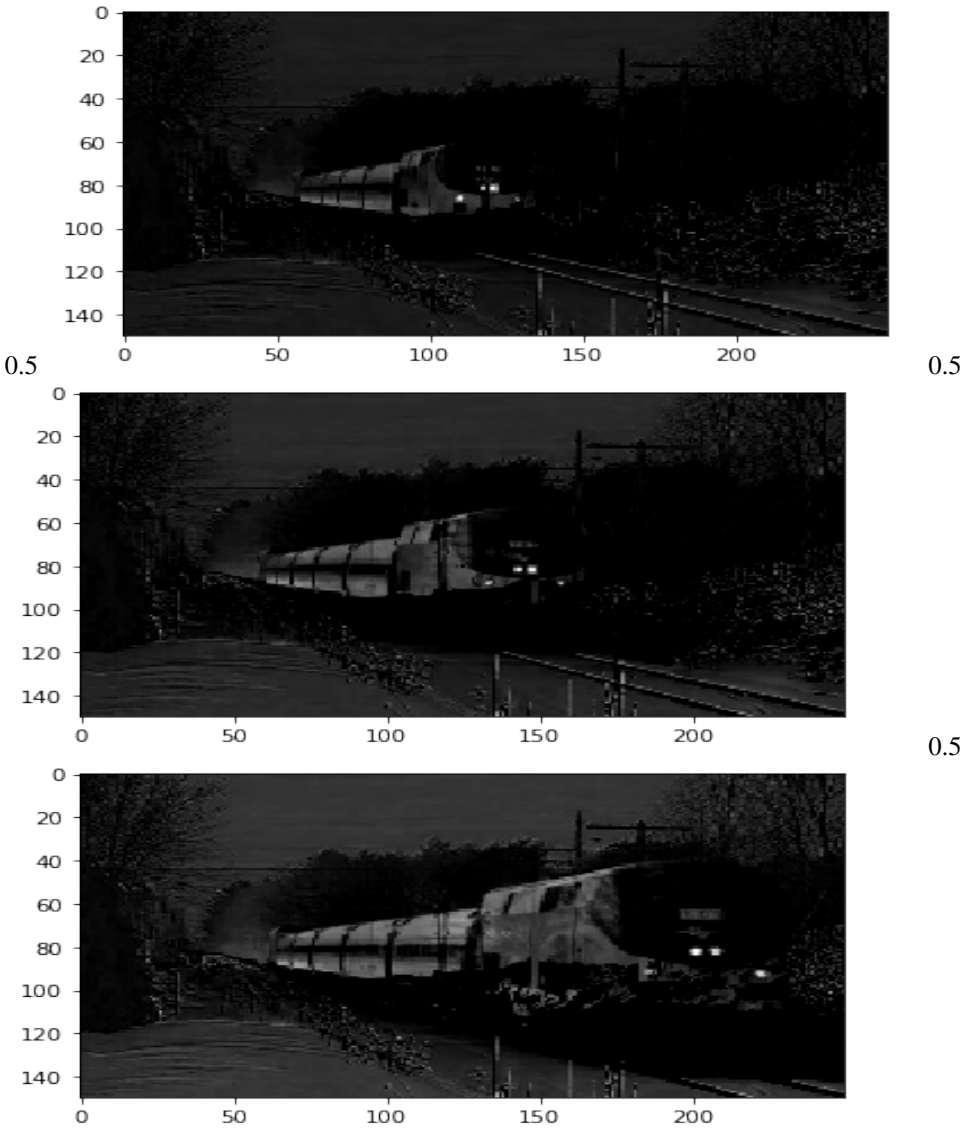


Figure 3. The following pictures are the captures of the X-sparse matrix, which represents the foreground

Figure 1 is the plot of the combination matrices of low-rank matrix and sparse matrix, instead of simply plotting a data matrix. However, result was just like a plotting of data matrix. It clearly shows the background and the movement of trains in the foreground.

From figure 2, it shows the capture of background. Although background should only consist of the scenery except the moving train in the foreground, the movement of train is visible in our plot. However, if you take a look at it carefully, the scenery behind the train is visible, which makes the train transparent. It shows that our omega capture a little more

than just a background. This couldn't be fixed however, since it was just one omega that we chose for the background, so we couldn't possibly choose less.

Figure 3 lastly, shows the foreground. It successfully captures the movement of trains and the background is displays as dark. Yet, similar to our background, our foreground captures some parts of background such that stationary parts of the video such as rail road and tree was also captured and thus, visible.

Overall, the segmentation of video into foreground and background using DMD method was successful.

5 Summary and Conclusions

The application of Dynamic Mode Decomposition in segmenting videos frames into background and foreground has been introduced in the paper and the theory behind DMD algorithms has also been explained step-by-step. Although there were some limitations in the application, such that DMD algorithms was not able to perfectly separate backgrounds from foregrounds, the fact that DMD is equation-free, data-based algorithms definitely makes DMD attractive to use for many other data science applications. The following application was so powerful such that we were able to analyze without any other given data, other than a single video. We believe DMD method has lots of potential and many other applications in real life.

6 Appendix

Link to the a python file and a html file:
`https://drive.google.com/open?id=1fmgabwpYUbME2QTHo8DlCAQWMAr420Em`

6.1 Jupyter Notebook Codes

In [1]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt

from skimage.transform import resize
```

In [5]:
```
def videoProcessor(path):
    cap = cv2.VideoCapture(path)
    fps = int(cap.get(cv2.CAP_PROP_FPS))

    frames = []
    # Read until video is completed
    while(cap.isOpened()):
        ret, frame = cap.read()

        if ret == True:
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            gray = resize(gray, [150, 250])
            frames.append(np.reshape(gray, (150*250, 1)))
        else:
            break

    # each image (frame) is represented as a column
    matrix = np.hstack(frames)
    # create DMD input-output matrices
    X1 = matrix[:, :-1]
    X2 = matrix[:, 1:]
    return matrix, X1, X2, fps
```

In [3]:
```
def DMDdecomp(X1, X2, fps):
    U, S, Vh = np.linalg.svd(X1, full_matrices=False) #reduced SVD
    # Differences with matlab
    # 1. S is a 1d array of diagonal elements
    # 2. Vh = V': the matlab version returns V for X = U S V',
    #    whereas python returns V'

    # truncate zero values from svd
    r = np.linalg.matrix_rank(X1)
    Ur = U[:, :r]
    Sr = np.diag(S)[:r, :r]
    Vr = Vh.conj().T[:, :r]

    # build A tilde
    # Ur'*X2*Vr*inv(Sr)
    Atilde = np.dot(np.dot(np.dot(Ur.conj().T, X2), Vr), np.linalg.inv(Sr))  # r by r
    # eigenvalue decomposition
    mu, W = np.linalg.eig(Atilde)
    # X2*Vr*inv(Sr)*W
    Phi = np.dot(np.dot(np.dot(X2, Vr), np.linalg.inv(Sr)), W)  # DMD Modes

    dt = 1/fps
    omega = np.log(mu)/dt

    return r, omega, Phi
```

In [6]:
```
X, X1, X2, fps = videoProcessor("train.mp4")
print("There are", fps, "frames per sec")
print("The length of the video:", round(len(X[0])/fps), "sec")
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/skimage/transform/_warps.py:84: UserWarning: The default mode, 'constant', will be changed to 'reflect' in skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "

There are 29 frames per sec
The length of the video: 25 sec
```

In [23]:
```
r, omega, Phi = DMDdecomp(X1, X2, fps)
bg = np.flatnonzero(np.abs(omega)<1e-02)
fg = np.setdiff1d(np.arange(1,r), bg)
print("rank:", r)
print("The number of background omegas:", len(bg))
print("The number of foreground omegas:", len(fg))
```

```
rank: 719
The number of background omegas: 1
The number of foreground omegas: 717
```

In [8]:
```
omega_fg = omega[fg]
Phi_fg = Phi[:,fg]
omega_bg = omega[bg]
Phi_bg = Phi[:,bg]
```

In [9]:
```
# Compute DMD Background solution
b = np.dot(np.linalg.pinv(Phi), X[:, 1]) # initial amplitudes for the modes.
t = np.arange(len(X[0]))*(1/fps) # time vector
X_____bg = np.zeros((np.size(omega_____bg), len(t)))
for tt in range(len(t)):
    X_____bg[:,tt] = b[bg]*np.exp(omega_____bg * t[tt])
X_____bg = Phi_____bg*X_____bg
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/ipykernel_____launcher.py:6:
ComplexWarning: Casting complex values to real discards the imaginary part
```

We don't really need this code though.

In [81]:
```
# # Compute DMD Foreground solution
# b = np.dot(np.linalg.pinv(Phi), X[:, 1]) # initial amplitudes for the modes.
# t = np.arange(len(X[0]))*(1/fps) # time vector
# X_____fg = np.zeros((np.size(omega_____fg), len(t)))
# for tt in range(len(b)):
#     X_____fg[:,tt] = np.exp(omega_____fg * t[tt]) * b[fg]
# X_____fg = np.matmul(Phi_____fg, X_____fg)
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/ipykernel_____launcher.py:6:
ComplexWarning: Casting complex values to real discards the imaginary part
```

In [10]:
```
X_____lowrank = X_____bg
X_____sparse = X - np.abs(X_____lowrank)
R = np.where(X_____sparse<0, X_____sparse, 0)
X_____lowrank = R + np.abs(X_____lowrank)
X_____sparse = X_____sparse - R
```

In [19]:
```
for i in range(X.shape[1]):
    x = np.reshape(X_____sparse[:,i], (150, 250))
    plt.imshow(x, cmap="gray")
    plt.pause(0.5)
```

References

[1] ”Dynamic mode decomposition - data-driven modeling of complex systems” by J. Nathan Kutz et al.

[2] aaren. Sparse DMD in Python. GitHub, 21 Dec 2015, `github.com/aaren/sparse_dmd`.

[3] Ianscape. Test HD Stationary. YouTube, YouTube, 4 Dec. 2008, `www.youtube.com/watch?v=WVJr0YHNNok`.