Motion Detection as a Consequence of Foreground/Background Separation using QR Streaming Dynamic Mode Decomposition

Peter Oehme*

23 April 2024

Abstract

This report discusses a QR compressed streaming DMD algorithm with applications to motion detection and foreground/background separation in real-time data-based video processing without full data availability. We explain how the iterative updating works hand in hand with the data compression to handle individually provided data and adaptively select relevant information within the video frames. Further, we indicate how a basic foreground/background separation algorithm directly yields a motion detection method in the streaming data setting, and how it can be adapted to serve its original purpose of foreground object detection. Three different realistic datasets are employed to demonstrate the efficacy of this approach, and to demonstrate its short-comings in noise resistance and background permanence.

1 Introduction

The Dynamic Mode Decomposition (DMD) currently enjoys a lot of attention for applications in data-driven settings due to its versatility. Most of these applications use the fact that DMD does not require any knowledge of the data to obtain generalised image processing systems ([11, 16, 2, 7, 21]). Among these image processing problems we tackle the tasks of foreground/background separation, and, due to the details of our implementation, motion detection. We demonstrate that our algorithms produce usable and reproducable results in a real-time viable runtime through the means of a streaming DMD implementation.

The two main aspects of streaming DMD are: the compression of the data, and the use of the streaming framework, where the algorithm operates on iteratively supplied data snapshots. The former of these two has made advances through the use of compressed sensing as well as randomised sampling methods, see e.g. [3] and [9]; however, the compression heavily depends on how well one can approximate currently available data, and most update computations are expensive. To remedy the update costs, a streaming DMD and an online DMD have been developped by [13] and [18]. Here, the authors successively update approximates of the discretised Koopman operator by updating either projection matrices or the underlying SVD directly. This report aims to lay out a third option to these two publications, one that does not change the main DMD algorithm, but instead uses update formulae of the QR decomposition and the SVD to keep track of the necessary quantities for which we compute the DMD as motivated by the examination of snapshot reconstruction in [6].

We structure this report as follows: Section 2 introduces the DMD and its accompanying quantities related to snapshot reconstruction and dynamic mode residuals. We also explain how simple QR compression works, and how it speeds up the evaluation of both the residuals and the reconstruction coefficients, concluding in an explanation of how to use update formulas to use QR decompression in an incremental algorithm. In Section 3 we highlight how foreground/background separation works once one has computed the DMD of data matrices. By analysis of our streaming and compressed algorithm we explain how the dynamically reconstructed backgrounds suit themselves to motion detection, and we provide a way to take some of these dynamic backgrounds to perform the original separation task instead of motion detection. We demonstrate the performance of our basic Python implementation on three sample datasets in Section 4, and conclude the report in Section 5 by pointing out various ways of further research to improve the results presented here.

*EPFL, Lausanne, Switzerland (peter.oehme@epfl.ch)

2 Dynamic Mode Decomposition

As a crucial method for this report we use the *Dynamic Mode Decomposition* (DMD). DMD is a data-agnostic computational tool founded on Koopman operator theory [19, 22]. For a discrete dynamical system

$$x_{k+1} = Kx_k,$$

where K is an unknown linear operator, DMD uses an algorithm such as Algorithm 1 to compute approximate eigenpairs known as dynamic modes and dynamic amplitudes, or Ritz pairs, of K from a sequence of data snapshots $(\mathbf{f}_k)_{k=1}^{M}$, $M \in \mathbb{N}$, split into two data matrices $\mathbf{X} = (\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{M-1})$ and $\mathbf{Y} = (\mathbf{f}_2, \mathbf{f}_3, \ldots, \mathbf{f}_M)$, by computing Galerkin approximations of the original matrix. Afterwards, we can use the dynamic pairs to compute reconstructions of the original data and make predictions for future points in time based on these reconstruction.

Algorithm 1: Schmid DMD, introduced in [20]

Data: Data matrices $\mathbf{X}, \mathbf{Y} \in \mathbb{C}^{N\times M}$, an optional tolerance $\tau > 0$

1 Compute the SVD $\mathbf{X} = U\Sigma V^*$, truncate the SVD to dimension R for a given tolerance τ;
2 Define the approximate $\tilde{K} = U^*\mathbf{Y}V\Sigma^{-1} \in \mathbb{C}^{R\times R}$;
3 Determine R eigenpairs $(\lambda_i, w_i), i = 1, 2, \ldots, R$;
4 Set $W = (w_1, w_2, \ldots, w_R)$;
5 **return** *Ritz values* $\lambda_1, \lambda_2, \ldots, \lambda_R$ *and Ritz vectors* UW

Unfortunately, not all dynamic modes are good approximations of K's eigenvalues on the data matrices $\mathbf{X}$ and $\mathbf{Y}$. To get a feeling for how well each DMD mode can approximate an eigenpair of K we consider the *data-driven Ritz residuals* of the dynamic modes $Z = UW$ defined in [5] as follows:

$$r_i \coloneqq \|K(Uw_i) - \lambda_i Uw_i\|_2 = \|(\mathbf{Y}V\Sigma^{-1})w_i - \lambda_i z_i\|_2. \tag{1}$$

We use these residuals not only to check for the suitability as eigenpairs, but they can also give us an indication for other properties we shall discuss in Section 3.

After selecting a subset $\mathcal{I} \subseteq \{1, 2, \ldots, R\}$ with $\ell \in \mathbb{N}$ indices of the available dynamic modes, we want to reconstruct the known data from these modes. This is equivalent to solving the problem

$$\min_{\alpha\in\mathbb{C}^\ell} \sum_{i=1}^{M-1} \omega_i \left\| \mathbf{f}_i^2 - \sum_{j=1}^{\ell} \alpha_{\mathcal{I}_j} \lambda_{\mathcal{I}_j}^{i-1} z_{\mathcal{I}_j} \right\|_2^2 = \min_{\alpha\in\mathbb{C}^\ell} \|(\mathbf{X} - Z\,\mathrm{diag}\,(\alpha)\,\mathbb{V}(\lambda))W\|_F^2$$

as defined in [6, Equation (3.1)], where $\mathbb{V}(\lambda) \in \mathbb{C}^{\ell\times M-1}$ is the Vandermonde matrix of the dynamic amplitudes and $\omega_i \in \mathbb{C}$ are given weights. Following the argument in [6], the solution to this least squares problem is given by

$$\alpha = \left((Z^*Z) \odot (\overline{\mathbb{V}(\lambda)W^2\mathbb{V}^*(\lambda)})\right)^{-1} \left(\overline{\mathbb{V}(\lambda)W} \odot (Z^*\mathbf{X}W)e\right), \tag{2}$$

where e is the vector containing only ones, and $\odot$ denotes the Hadamard product of matrices. Finally, we define the reconstruction of a snapshot at time k as the combination of this parameter vector with the other, previously computed quantities:

$$\mathbf{x}_{\mathrm{re}}(k) \coloneqq \sum_{i=1}^{\ell} \alpha_{\mathcal{I}_i} \lambda_{\mathcal{I}_i}^{k-1} z_{\mathcal{I}_i}. \tag{3}$$

We will use this reconstruction formula repeatedly and for different ways of computing the coefficients α for the same input data. Most importantly we note that by splitting the indices over which we sum further into two sets $\mathcal{I}_1$ and $\mathcal{I}_2$, we obtain a decomposition $\mathbf{x}_{\mathrm{re}} = \mathbf{x}_{\mathrm{re}}^{(1)} + \mathbf{x}_{\mathrm{re}}^{(2)}$, which we will use in Section 3 after consideration of a proper splitting method.

2.1 QR Compressed DMD

Another option of getting around the prohibitively large dimension N is to compress the data before computing any DMD quantities. There exist multiple frameworks for these compressions, such as Compressive Sensing [3],

QR Compression [6], and Randomised Sampling [9]. In this report we choose compression by means of the QR decomposition $QR = (\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_M)$. As a consequence, we introduce two layers of abstraction: the uncompressed and the compressed. We can classify the general steps of algorithms relying on DMD as belonging to one of these layers as follows:

1. In the uncompressed layer we first have to compress the data $(\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_M) = QR$,
2. then we perform DMD on the compressed data obtained from the columns of R, and
3. lastly we project reconstructions and DMD modes back up into the uncompressed layer for analysis and further treatment as part of our general data processing objective.

Most of the previous quantities can be computed in the compressed domain, thus allowing for more efficient computation overall if we do not require any access to the full dimensional quantities.

Importantly, we can compute the residuals of the dynamic pairs as

$$r_i = \|\mathbf{Y}V\Sigma^{-1}w_i - \lambda_i QUw_i\|_2 = \|QR(:, 2\colon M)V\Sigma^{-1}w_i - \lambda_i QUw_i\|_2 = \|R(:, 2\colon M)V\Sigma^{-1}w_i - \lambda_i Uw_i\|_2,$$

assuming that we calculated $R = U\Sigma V$ instead of the SVD of the uncompressed data, as well as the coefficients of the reconstruction through the following expression

$$\alpha = \left((R^*R) \odot (\overline{\mathbb{V}(\lambda)W^2\mathbb{V}^*(\lambda)})\right)^{-1} \left(\overline{\mathbb{V}(\lambda)W} \odot (R^*\mathbf{G}W)e\right),$$

where $\mathbf{G} = Q^*\mathbf{X}$ are the compressed data snapshots, where the structure remains similar to the original problem in Equation (2). On the other hand, to obtain the uncompressed reconstructions and dynamic modes we always need to compute $Q\mathbf{f}_{\text{re}}$ or Qz_i, which is a disadvantage only if we need frequent access to the high-dimensional modes.

2.2 QR Streaming DMD

To our benefit, the QR decompression introduced in Subsection 2.1 can be easily updated to a streaming setting: Instead of having all data snapshots available at once, we get each $\mathbf{f}_k$ one by one. As such, it is necessary to devise a way to continuously update the compressed representation of our available data. We do this in two steps: First off all, it is necessary to add columns to the QR decomposition. This is generally useful in the streaming setting because we always assume to get more data, one at a time. Secondly, in the case of a limited snapshot memory we want to delete the first column of the present QR decomposition to save memory in the long run and to speed up the individual computations we might undertake in the compressed domain.

Adding a column to an already existing QR decomposition works as follows: Suppose your current data matrix is $\mathbf{X}_{\text{old}} = (\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{k})$ and we know its QR decomposition $\mathbf{X}_{\text{old}} = QR$. To compute the QR decomposition of the new data matrix $(\mathbf{X}_{\text{old}}, \mathbf{f}_{k+1})$, we require the QR decomposition of the incoming data snapshot orthogonalised w.r.t. the previous QR basis

$$(\text{id} - QQ^*)\mathbf{f}_{k+1} = \tilde{Q}\tilde{R}.$$

This allows us to write $\mathbf{X}_{\text{new}}$ as the updated QR decomposition

$$\mathbf{X}_{\text{new}} = (Q, \tilde{Q}) \begin{pmatrix} R & Q^*\mathbf{f}_{k+1} \\ 0 & \tilde{R} \end{pmatrix}.$$

Removing the first column, on the other hand, is a bit more involved. In this case, we desire to know the QR decomposition of $\mathbf{X}_{\text{new}} = (\mathbf{f}_2, \mathbf{f}_3, \ldots, \mathbf{f}_k)$, whereas we a priori know only $QR = \mathbf{X}_{\text{old}} = (\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_k)$. Simply removing the first column of R is not sufficient because the new matrix $\tilde{R}$ would not be triangular, instead we observe the following sparsity pattern:

$$\tilde{R} = \begin{pmatrix} \times & \times & \ldots & \times \\ \times & \times & \ldots & \times \\ & \times & \ldots & \times \\ & & \ddots & \vdots \\ \mathbf{0} & & & \times \end{pmatrix}$$

Thus, we need to rediagonalise this matrix. We use Givens or Householder rotations akin to [4] and obtain a sequence $H_{k-1} \dots H_2 H_1$ which, when applied to $\tilde{R}$, reduces it to upper triangular form. We gather the corresponding inverse rotations $H_1^* H_2^* \dots H_{k-1}^*$ and multiply all of them before applying them to the Q matrix, thus resulting in

$$Q\tilde{R} = \underbrace{Q(H_1^* H_2^* \dots H_{k-1}^*)}_{=:\hat{Q}} \underbrace{(H_{k-1} \dots H_2 H_1)\tilde{R}}_{=:\hat{R}}.$$

Finally, we are left with the new QR decomposition $\mathbf{X}_{\text{new}} = \hat{Q}\hat{R}$.

A prototype of a streaming DMD implementation for sequential data, that is $\mathbf{y}_k = \mathbf{x}_{k+1}$, is given in Algorithm 2. The parameter $\ell_{\text{mem}} \in \mathbb{N}$ represents the length of the snapshot memory $\mathfrak{M}$, that is the maximum number of data snapshots stored throughout the runtime of the algorithm. Initially, we set the snapshot memory $\mathfrak{M}$ as the empty set.

Algorithm 2: QR Streaming DMD

Data: Incoming data $\mathbf{x}_k, \mathbf{y}_k \in \mathbb{C}^N$, the current memory $\mathfrak{M}$, the maximal size of memory ℓ_{mem}, an optional tolerance $\tau > 0$

if $|\mathfrak{M}| = 0$ **then**
  Set $\mathfrak{M} = \{\mathbf{x}_k\}$;
  Compute the QR decomposition $QR = \mathtt{qr}((\mathfrak{M}, \mathbf{y}_k))$;
  Compute the SVD $R(:,1) = U\Sigma V^*$;
else if $|\mathfrak{M}| > 0$ *and* $|\mathfrak{M}| < \ell_{mem}$ **then**
  Append $\mathbf{x}_k$ to $\mathfrak{M}$;
  Update the QR decomposition by adding the column $\mathbf{y}_k$;
  Update the SVD decomposition by adding the column $\mathbf{y}_k$;
else
  Append $\mathbf{x}_k$ to $\mathfrak{M}$, remove the first element of $\mathfrak{M}$;
  Update the QR decomposition by adding the column $\mathbf{y}_k$ and removing the first column;
  Update the SVD decomposition by adding the column $\mathbf{y}_k$ and removing the first column;
return *Schmid DMD on* $R(:,1{:}\,|\mathfrak{M}|), R(:,2{:}\,\mathit{end})$ *with tolerance* τ*, see Algorithm 1*

2.3 Updating the SVD

After every update of the QR decomposition, be it the addition of a column, the deletion of the first column, or both during a single iteration of the streaming algorithm, we are left with a new matrix R. During the QR compressed DMD algorithm, we would need to recalculate the SVD of this matrix after every update, however we can use similar updating procedures for the SVD. Suppose that we have previously computed the SVD $\mathbf{X}_{\text{old}} = U_{\text{old}}\Sigma_{\text{old}}V_{\text{old}}^* \in \mathbb{C}^{N\times k}$. The addition of a column begins with the following sequence of computations:

$$\begin{aligned}
\mathbf{X}_{\text{new}} &= (\mathbf{X}_{\text{old}}, 0) + \mathbf{f}_{\text{new}} e_{k+1}^T \\
&= U_{\text{old}}\Sigma_{\text{old}}(V_{\text{old}}^T, 0) + \mathbf{f}_{\text{new}} e_{k+1}^T \\
&= (U_{\text{old}}, \mathbf{f}_{\text{new}}) \begin{pmatrix} \Sigma_{\text{old}} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} V_{\text{old}}^* & 0 \\ 0 & 1 \end{pmatrix} \\
&= (U_{\text{old}}, \mathbf{r}) \begin{pmatrix} \Sigma_{\text{old}} & \mathbf{d} \\ 0 & \rho \end{pmatrix} \begin{pmatrix} V_{\text{old}}^* & 0 \\ 0 & 1 \end{pmatrix},
\end{aligned}$$

where we used the following projected quantities:

$$\mathbf{d} \coloneqq U^*\mathbf{f}_{\text{new}}, \quad \mathbf{p} \coloneqq (\mathrm{id} - UU^*)\mathbf{f}_{\text{new}}, \quad \rho \coloneqq \|\mathbf{p}\|_2, \quad \mathbf{r} = \mathbf{p}/\rho.$$

As a last step, it remains to compute the SVD of the central arrowhead matrix

$$\begin{pmatrix} \Sigma_{\text{old}} & \mathbf{d} \\ 0 & \rho \end{pmatrix} = \tilde{U}\tilde{\Sigma}\tilde{V}^*,$$

for which there exist efficient algorithms with a runtime of $\mathcal{O}((k+1)^2)$, see e.g. [12, 14, 15]. Afterwards, we assemble the new SVD by multiplying

$$\mathbf{X}_{\text{new}} = ((U_{\text{old}}, \mathbf{r})\tilde{U})\tilde{\Sigma}\left(\tilde{V}^* \begin{pmatrix} V_{\text{old}}^* & 0 \\ 0 & 1 \end{pmatrix}\right) = U_{\text{new}}\Sigma_{\text{new}}V_{\text{new}}^*.$$

Thus, whenever we add a row to the data matrix and update the corresponding QR decomposition, we can similarly update the underlying SVD for a faster computation.

The analogous removal of the first column also reduces to the computation of the eigendecomposition of a diagonal-plus-rank-one matrix, which for brevity we leave out of this report. If the reader is interested in these methods, we refer them to sources such as [15].

3 Foreground/Background Separation and Motion Detection

In this section, we introduce an application of DMD in image processing. In particular, we link the Ritz values computed in Section 2 to a segmentation of foreground and background data, and use these decompositions to compute a priori data-agnostic foreground masks. This approach was first introduced in [11], where the authors computed the DMD for a set of video frames and reconstructed one background snapshot to compute a foreground mask for every frame. This paper relies on the full decomposition, making it slow when compared to different compressed approaches such as compressive DMD or randomised DMD.

3.1 Foreground/Background Separation

The main idea used in [11, 16, 8] is to compare the logarithmic dynamic amplitudes

$$\omega_i = \log(\lambda_i), \quad i = 1, 2, \ldots, R,$$

and find those pairs for which the logarithmic amplitude is close to zero. These Ritz values change very slowly during each successive time step in the reconstruction 3, hence it is suitable to consider these as the background of the data. Setting $\mathcal{I}_{\text{BG}} \subseteq \{1, 2, \ldots, R\}$ as the subset of these background amplitudes we can split the reconstruction formula into a foreground and a background object as follows

$$\mathbf{f}_{\text{re}}(k) = \mathbf{f}_{\text{BG}}(k) + \mathbf{f}_{\text{FG}}(k) = \sum_{i \in \mathcal{I}_{\text{BG}}} \alpha_i \lambda_i^{k-1} z_i + \sum_{i \notin \mathcal{I}_{\text{BG}}} \alpha_i \lambda_i^{k-1} z_i.$$

Lastly, we define the foregound mask as the thresholded matrix-valued function

$$(m_{\text{FG}}(k))_{i,j} \coloneqq \begin{cases} 1, & |\mathbf{f}_k - \mathbf{f}_{\text{BG}}(k)| \geq \tau \\ 0, & |\mathbf{f}_k - \mathbf{f}_{\text{BG}}(k)| < \tau \end{cases},$$

where $\tau > 0$ is a tolerance chosen prior to the computation.

3.2 Backgrounds in QR Streaming DMD

The procedure described in Subsection 3.1 illustrates the application of the DMD method to imaging problems, given that we know the entirety of the data matrix beforehand. This assumption may not be valid in cases where storing large amounts of data is impractical or the data are not available all at once. In such a case it is convenient to employ a streaming framework and iteratively compute the desired results.

There is, however, one immediate problem: If we constrain the size of the memory available during the computation, we might encounter the case that all relevant data of the actual background does no longer appear in the kept memory. Consider, for example, a streaming implementation with a memory of exactly 10 snapshots. Suppose that $\mathbf{f}_T$ contains only the background, and that every snapshot after it has some moving object in it. Then, after advancing 10 steps in time, the memory of the method is $\{\mathbf{f}_{T+1}, \mathbf{f}_{T+2}, \ldots, \mathbf{f}_{T+10}\}$, and therefore no longer contains any pure representation of the original background. Thus, a comparison of the streaming background with the incoming data yields a foreground mask only containing local and recent changes: that is if some object moves

within a few frames of the video, then we can detect changes in its position. Effectively, this method only allows for detection of a *motion mask* instead of complete foreground/background separation, where the motion is identified as the change with respect to the background recovered from the most recent streaming data. A side effect of this analysis is that large moving objects with a uniform texture will register in the motion mask only along the moving edges, e.g. the front and the back of a moving bicycle. We illustrate this behaviour more in Section 4.

To mitigate the lack of a fixed reference background for streaming applications with limited available memory, we can try and compute a more permanent background by evaluating the approximation quality of the current DMD residuals as defined in Equation 1. Looking at the minimum absolute value of the residuals from the numerical example in Subsection 4.1, we notice that at some points in time the value increases significantly, whereas before it remained about constant, see Figure 1. We exploit this by updating the stored background reconstruction whenever the minimal residual is below a certain predefined tolerance $\rho > 0$. Whenever this threshold is exceeded, we compute the foreground mask of the current frame from the difference $\mathbf{f}_k - \mathbf{f}_{\mathrm{BG}}$, where $\mathbf{f}_{\mathrm{BG}}$ is the most recent background reconstruction with a minimal residual smaller than the tolerance. This has the additional benefit that we do not compute a foreground mask for pure background snapshots, hence reducing the output of the algorithm to actual time periods of interest, that is spans of time in which foreground objects move.

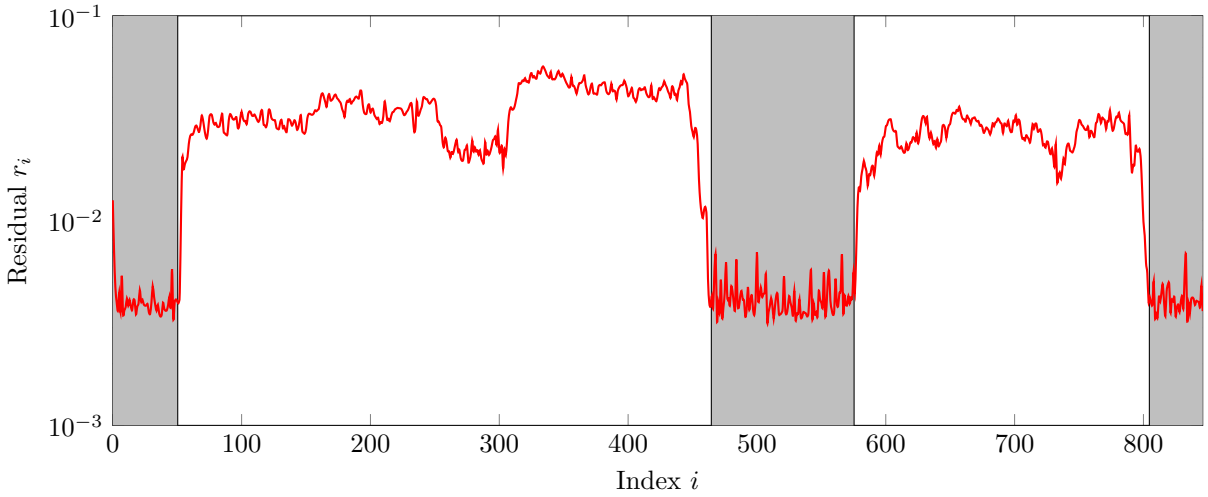


Figure 1: Residuals r_i for the pedestrian dataset from Subsection 4.1 for a memory size of 5 data snapshots; Red: residuals as computed during the streaming algorithm; Grey shaded areas indicate indices of frames where the algorithm determined the residual to be less than the specified tolerance ρ.

This method has the benefit that backgrounds can change continually throughout the stream of data snapshots, and after a short readjustment period a relevant background will be computed. In contrast, a non-streaming DMD method relies entirely on the accuracy of the background prediction from all data at once, and may be substantially worse. Thus, using sets of data limited in time not only speeds up every single evaluation and allows for real-time applications, but it may also provide a more stable method.

4 Numerical Experiments

Throughout this section we present some numerical experiments. These were selected from the changedetection.net datasets. Our procedure works as follows: While reading the data stream of images we normalise every image vector by its norm, such that $\|\mathbf{f}_k\|_2 = 1$ holds for all input data. This data normalisation is useful in DMD because the scaling of the data directly influences the numerical computations and thus the quality of the returned quantities [5]. Afterwards, we perform QR compressed streaming DMD as explained in Algorithm 2 on the incoming data. In a first run we plot the minimal values of all dynamic residuals for a small number of data snapshots to decide upon a cut-off tolerance for background and motion detection. Once this threshold has been chosen, we compute the background reconstructions and foreground masks as explained in Section 3. We repeat this process for a select number of parameter combinations, mostly varying the size of the data memory ℓ_{mem}. Our experiments show that in the streaming setting even a small number of stored snapshots and a low reconstruction/truncation rank suffice to provide good foreground masks.

All code used to generate the results of this report has been uploaded to github.com/peoe/dmd-math-656/. Included in this repository are a setup file for the Python environment required, as well as a script to aid in acquiring the relevant datasets.

4.1 Detection of Pedestrians

The pedestrian dataset obtained from changedetection.net shows a section of a street with pedestrians and a bicyclist passing along the road. This example is a nice benchmark problem because the video contains both a stationary and not very noisy background, but it also includes different movement speeds.

We run both the motion detection and the foreground/background separation algorithm in the streaming DMD setting with a limited snapshot memory size of $\ell_{\text{mem}} \in \{5, 10, 15, 20\}$. For the foreground/background separation we consider the minimal residuals of the first 100 frames to determine the residual tolerance $\rho > 0$, see Figure 2 for a plot of these first residuals. Afterwards, we compute the separated foreground data with tolerances of $\rho \in \left\{8 \cdot 10^{-3}, 5 \cdot 10^{-3}, 3 \cdot 10^{-3}, 10^{-3}\right\}$ for the respective ℓ_{mem} snapshot memory size. We can see that the minimal residual for every memory size decreases initially as the memory slowly gets filled with snapshots. Then we enter a plateau phase where the minimal residual remains consistently small before increasing after $i = 50$ where the first pedestrian enters the frame. Noticably, the minimal residual for larger memory sizes spikes to a large number as soon as the scene changes upon the entrance of a moving foreground object. This is due to the fact that at this point the snapshot memory contains a larger number of snapshots do not contain the pedestrian, and hence the dynamic modes mostly reconstruct the static background instead of the moving foreground. This can be mitigated by choosing either a smaller memory size ℓ_{mem}, or by dynamically reducing the length of the memory once we detect a spike in the minimal residual.

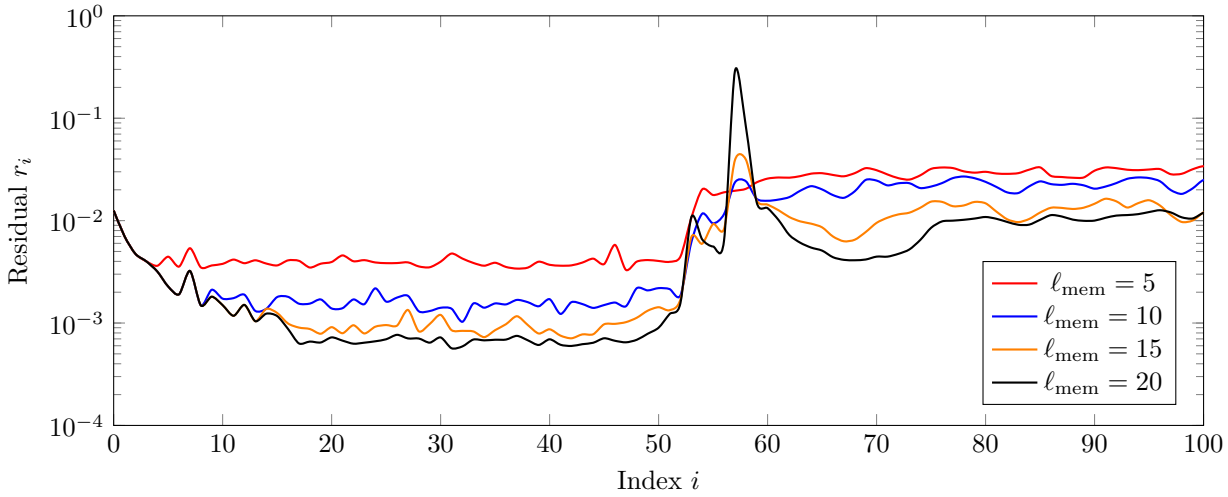
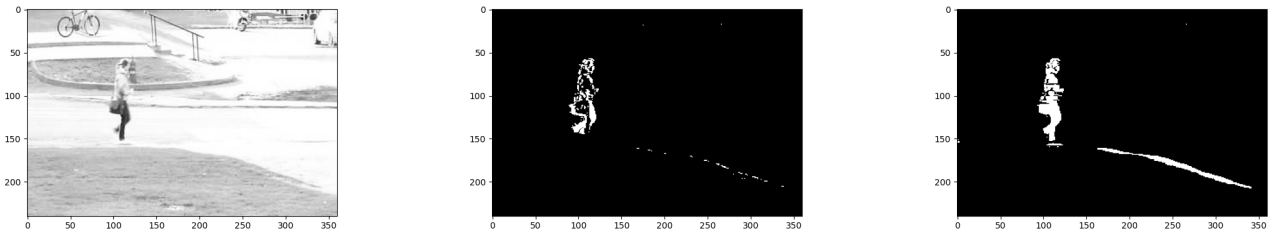


Figure 2: Residuals r_i for the pedestrian dataset for memory sizes $\ell_{\text{mem}} \in \{5, 10, 15, 20\}$; In colour: residuals as computed during the streaming algorithm.

For selecting the motion or foreground masks we determine an error threshold and select all pixels for which the dynamic background differs from the incoming data by more than the threshold. A comparison of the true image, the computed motion mask, and the foreground mask at frame number 100 can be seen in Figure 3. Notice that the motion mask highlights only the outline of the pedestrian — as described above this occurs because the dynamic backgrounds in the motion detection procedure only store local information, and thus only detect the movement along object and texture boundaries.

4.2 Detection of Moving Canoe

The canoe dataset obtained from changedetection.net shows a section of a river and trees with a canoe passing in front of the camera. In contrast to the pedestrian problem of Subsection 4.1, this video possesses a noisy background due to reflections and changes in shade of both the river and the trees. Additionally, most of the frame consists of dark green and brown tones, which means that the greyscaled frames do not exhibit large differences between foreground and background objects.



(a) True image (b) Motion mask (c) Foreground mask

Figure 3: Comparison of the true image, the motion mask, and the foreground mask for the pedestrian example at frame 100.

We repeat the same process as in the pedestrian example of Subsection 4.1 for the memory sizes $\ell_{\text{mem}} \in \{5, 10, 15, 20\}$. In this case we need to choose the tolerances for foreground detection in a more careful manner because the video is much more noisy due to reflections on the river. This can be seen in the much less significant difference in minimal residuals in Figure 4. After experimentation we decided to use the tolerances $\rho \in \left\{8 \cdot 10^{-2}, 4.4 \cdot 10^{-2}, 2.8 \cdot 10^{-2}, 2.1 \cdot 10^{-2}\right\}$ for thresholding the minimal residuals.

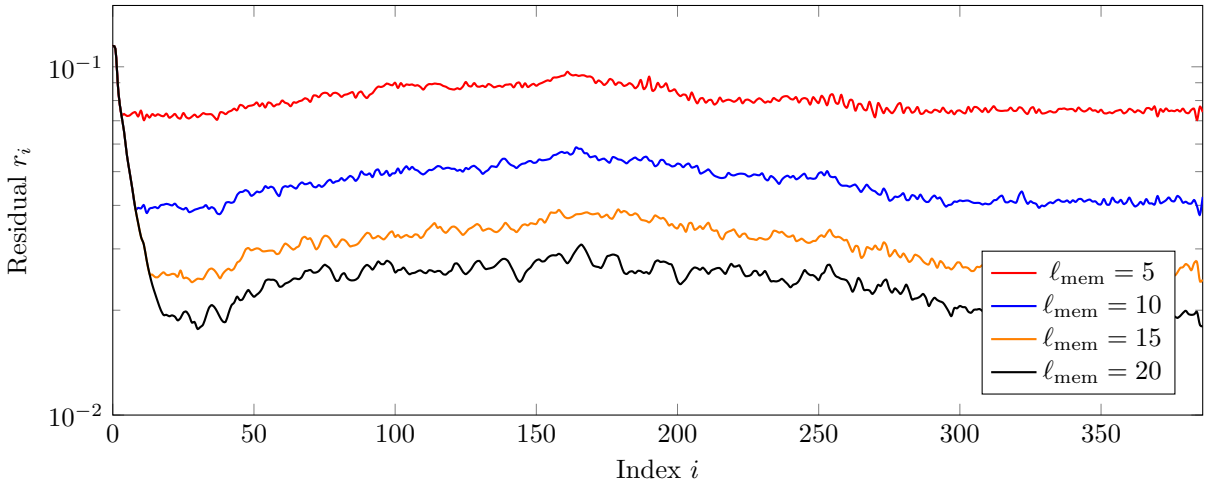


Figure 4: Residuals r_i for the canoe dataset for memory sizes $\ell_{\text{mem}} \in \{5, 10, 15, 20\}$; In colour: residuals as computed during the streaming algorithm.

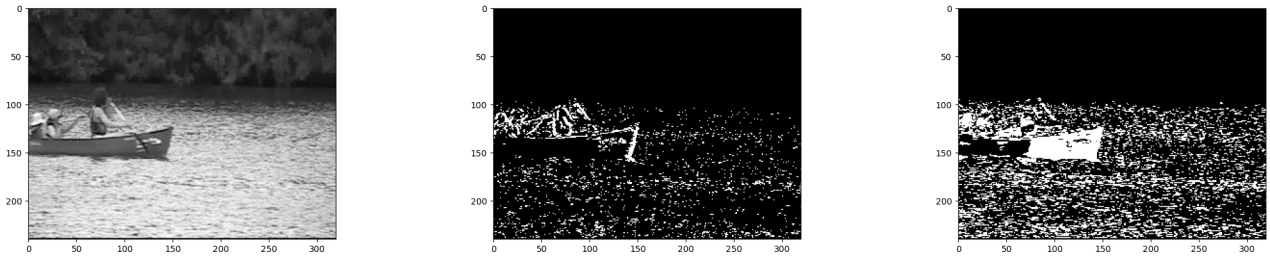
In the comparison of the motion and foreground masks we once again notice the outline effect of the motion detection algorithm. Unfortunately, both masks also suffer from noise pollution. The increased noise of the river reflections might be mitigated by using a regularized of denoised DMD strategy; however, we did not consider this line of argument further.

4.3 Detection of Sofa

The sofa dataset obtained from changedetection.net shows a sofa around which people move, sit down on the sofa, and handle cardboard boxes. This dataset is useful to consider because some foreground objects become part of the background, for example a box posed in front of the sofa or a person sitting down on the sofa turn into part of the localised background in the streaming DMD algorithm. This is in stark contrast to both previous examples, where the objects were clearly moving and did not stop for extended amounts of time.

This dataset does not have nearly as much noise as the canoe example from Subsection 4.2, which is why we choose the residual tolerances $\rho \in \left\{5 \cdot 10^{-3}, 2 \cdot 10^{-3}, 1.5 \cdot 10^{-3}, 10^{-3}\right\}$ for the corresponding memory sizes $\ell_{\text{mem}} \in \{5, 10, 15, 20\}$.

Similarly to the previous numerical experiments we notice that the motion mask highlights the outlines of moving foreground objects. Importantly, this example also highlights the disadvantages of the dynamic foreground



(a) True image (b) Motion mask (c) Foreground mask

Figure 5: Comparison of the true image, the motion mask, and the foreground mask for the canoe example at frame 100.

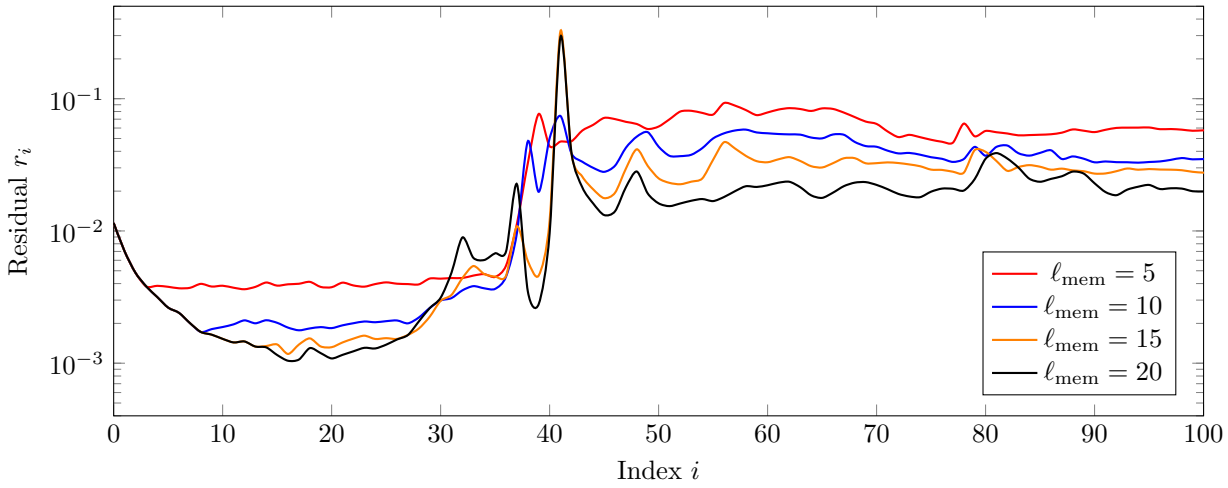
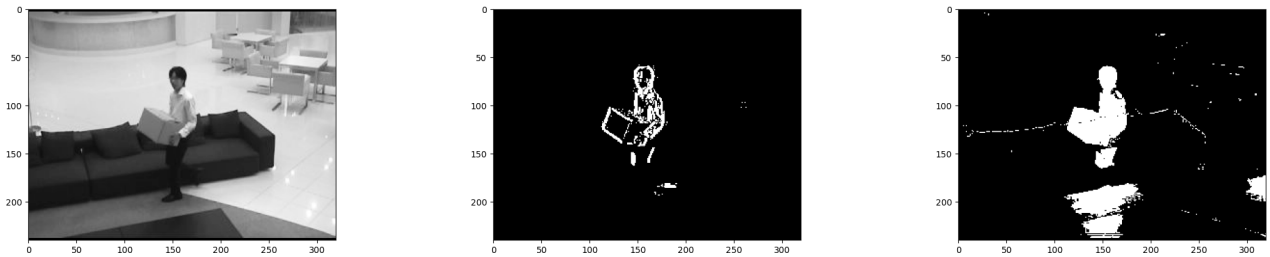


Figure 6: Residuals r_i for the sofa dataset for memory sizes $\ell_{\text{mem}} \in \{5, 10, 15, 20\}$; In colour: residuals as computed during the streaming algorithm.

detection algorithm: whenever the foreground objects come to rest the algorithm determines them to be part of the background, hence any prolonged resting may produce faulty results for the supposed foreground. A potential solution to this is to fix the dynamic background once it is established and not update it thereafter; however, this also harbours the downside that the video needs to start with a stationary "groundtruth" background, i.e. the dynamic algorithm becomes subject to priming in the input data.



(a) True image (b) Motion mask (c) Foreground mask

Figure 7: Comparison of the true image, the motion mask, and the foreground mask for the sofa example at frame 100.

5 Conclusion and Outlook

In this report we discuss a dynamic algorithm for motion detection and foreground/background separation in a data-agnostic setting using QR compressed streaming DMD. We demonstrate that the algorithm can produce usable and interpretable results independent of the data supplied after fine tuning of only a small number of parameters. The resulting method is significantly faster than classical DMD, where additionally all data needs to be accessible all at once. Thus, the ideas described in this report are suited in particular to real time applications, where both memory is limited and speed is of essential importance.

To add onto the ideas of this report, a number of approaches may be eligible. Instead of applying direct QR compression one could employ a randomised method such as Randomised QR decompositions, see e.g. [17, 1, 10]. Additionally, different compression regimes such as randomised DMD [9] or compressed sensing DMD [3] could be used, though it would be necessary to consider if these frameworks can be efficiently adapted to a streaming framework. Furthermore, this work can be expanded by using an adaptive extension approach such as the adaptive basis update of [13]. Another aspect to examine is the memory size — while a fixed size snapshot memory works remarkably well, it may be that an adaptively growing or shrinking memory provides further benefits. Finally, rigorous analysis of the tolerance hyperparameters would aid in tuning the algorithm on-the-fly to avoid potentially expensive precalculations, and usage of methods to increase noise resistance would improve the performance in more challenging data settings.

References

[1] O. Balabanov. *Randomized Cholesky QR Factorizations*. 2022. DOI: 10.48550/arXiv.2210.09953. arXiv: 2210.09953. preprint.

[2] C. Bi, Y. Yuan, J. Zhang, Y. Shi, Y. Xiang, Y. Wang, and R. Zhang. "Dynamic Mode Decomposition Based Video Shot Detection". In: *IEEE Access* 6 (2018), pp. 21397–21407. DOI: 10.1109/ACCESS.2018.2825106.

[3] S. L. Brunton, J. L. Proctor, J. H. Tu, and J. N. Kutz. "Compressed Sensing and Dynamic Mode Decomposition". In: *Journal of Computational Dynamics* 2.2 (2016), pp. 165–191. DOI: 10.3934/jcd.2015002.

[4] J. W. Daniel, W. B. Gragg, L. Kaufman, and G. W. Stewart. "Reorthogonalization and Stable Algorithms for Updating the Gram-Schmidt QR Factorization". In: *Mathematics of Computation* 30.136 (1976), pp. 772–795. DOI: 10.1090/S0025-5718-1976-0431641-8.

[5] Z. Drmač. "Dynamic Mode Decomposition—A Numerical Linear Algebra Perspective". In: *The Koopman Operator in Systems and Control: Concepts, Methodologies, and Applications*. Ed. by A. Mauroy, I. Mezić, and Y. Susuki. Lecture Notes in Control and Information Sciences. Cham: Springer International Publishing, 2020, pp. 161–194. DOI: 10.1007/978-3-030-35713-9_7.

[6] Z. Drmač, I. Mezić, and R. Mohr. "On Least Squares Problems with Certain Vandermonde–Khatri–Rao Structure with Applications to DMD". In: *SIAM Journal on Scientific Computing* 42.5 (2020), A3250–A3284. DOI: 10.1137/19M1288474.

[7] N. B. Erichson, S. L. Brunton, and J. N. Kutz. "Compressed Dynamic Mode Decomposition for Background Modeling". In: *Journal of Real-Time Image Processing* 16.5 (2019), pp. 1479–1492. DOI: 10.1007/s11554-016-0655-2. arXiv: 1512.04205.

[8] N. B. Erichson and C. Donovan. "Randomized Low-Rank Dynamic Mode Decomposition for Motion Detection". In: *Computer Vision and Image Understanding*. Award & Selection of Papers of ACCV 2014 146 (2016), pp. 40–50. DOI: 10.1016/j.cviu.2016.02.005.

[9] N. B. Erichson, L. Mathelin, J. N. Kutz, and S. L. Brunton. "Randomized Dynamic Mode Decomposition". In: *SIAM Journal on Applied Dynamical Systems* 18.4 (2019), pp. 1867–1891. DOI: 10.1137/18M1215013.

[10] L. Grigori and E. Timsit. "Randomized Householder QR". 2024.

[11] J. Grosek and J. N. Kutz. *Dynamic Mode Decomposition for Real-Time Background/Foreground Separation in Video*. 2014. DOI: 10.48550/arXiv.1404.7592. arXiv: 1404.7592. preprint.

[12] M. Gu and S. C. Eisenstat. "A Divide-and-Conquer Algorithm for the Bidiagonal SVD". In: *SIAM Journal on Matrix Analysis and Applications* 16.1 (1995), pp. 79–92. DOI: 10.1137/S0895479892242232.

[13] M. S. Hemati, M. O. Williams, and C. W. Rowley. "Dynamic Mode Decomposition for Large and Streaming Datasets". In: *Physics of Fluids* 26.11 (2014), p. 111701. DOI: 10.1063/1.4901016.

[14] N. Jakovčević Stor, I. Slapničar, and J. L. Barlow. "Accurate Eigenvalue Decomposition of Real Symmetric Arrowhead Matrices and Applications". In: *Linear Algebra and its Applications*. Special Issue on Eigenvalue Problems 464 (2015), pp. 62–89. DOI: 10.1016/j.laa.2013.10.007.

[15] H. Jiang and A. Chaudhuri. "A New Method to Improve the Efficiency and Accuracy of Incremental Singular Value Decomposition". In: *The Electronic Journal of Linear Algebra* 39 (2023), pp. 355–378. DOI: 10.13001/ela.2023.7325.

[16] J. N. Kutz, X. Fu, S. L. Brunton, and N. B. Erichson. "Multi-Resolution Dynamic Mode Decomposition for Foreground/Background Separation and Object Tracking". In: *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*. 2015 IEEE International Conference on Computer Vision Workshop (ICCVW). 2015, pp. 921–929. DOI: 10.1109/ICCVW.2015.122.

[17] P.-G. Martinsson, G. Quintana OrtÍ, N. Heavner, and R. van de Geijn. "Householder QR Factorization With Randomization for Column Pivoting (HQRRP)". In: *SIAM Journal on Scientific Computing* 39.2 (2017), pp. C96–C115. DOI: 10.1137/16M1081270.

[18] G. Nedzhibov. "Extended Online DMD and Weighted Modifications for Streaming Data Analysis". In: *Computation* 11.6 (6 2023), p. 114. DOI: 10.3390/computation11060114.

[19] C. W. Rowley, I. Mezić, S. Bagheri, P. Schlatter, and D. S. Henningson. "Spectral Analysis of Nonlinear Flows". In: *Journal of Fluid Mechanics* 641 (2009), pp. 115–127. DOI: 10.1017/S0022112009992059.

[20] P. J. Schmid. "Dynamic Mode Decomposition of Numerical and Experimental Data". In: *Journal of Fluid Mechanics* 656 (2010), pp. 5–28. DOI: 10.1017/S0022112010001217.

[21] I. Ul Haq, K. Fujii, and Y. Kawahara. "Dynamic Mode Decomposition via Dictionary Learning for Foreground Modeling in Videos". In: *Computer Vision and Image Understanding* 199 (2020), p. 103022. DOI: 10.1016/j.cviu.2020.103022.

[22] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley. "A Data–Driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition". In: *Journal of Nonlinear Science* 25.6 (2015), pp. 1307–1346. DOI: 10.1007/s00332-015-9258-5.