4
Fast rank and subspace tracking

In this chapter, we introduce FRAHST (*Fast Rank-Adaptive recursive row-Householder Subspace Tracking*), our extension that adapts both projection matrix and subspace rank on the fly. We provide details on how the original complexity of $\mathcal{O}(Nr + r^3)$ can be reduced to $\mathcal{O}(Nr + r^2)$. Subsequently, we show how the method is used for anomaly detection and how it can be integrated in a real-time monitoring solution.

4.1
Tracking the principal subspace basis

Strobach's algorithm presented in Section 3.3.4 is the state-of-the-art in the low complexity class subspace trackers but currently no adaptation has been put forward to cope with the rank tracking problem. Throughout our research, we experimented with several alternatives (such as YAST from Badeau et al. [2008] and FDPM from Doukopoulos and Moustakides [2008]) and found this particular method to have the best properties: provides excellent subspace estimates very efficiently in terms of computational and memory requirements, and has no special parameters to tune. The algorithm is an implementation of the orthogonal principle where the Q and S matrices (3-4) are directly updated through a recursive Householder reduction. During execution, the algorithm reads in new data snapshots $\boldsymbol{z}(t) \in \Re^N$ from N streams at time t and tracks the principal subspace projection matrix $\boldsymbol{Q}$ with dominant complexity of $3Nr$ flops, which is the lower bound for this class of algorithms where r is the *known and constant* subspace dimension.

We adopt the algorithm as listed in Figure 3.3 using the approximation of $\psi = 0$, consequently (3-23c) can be omited and (3-23d) simplified. We can save these few computation steps because we are only interested in subspace tracking and these simplifications are barely noticeable in practice (as seen in Figure 5.3) . This corresponds to the *a posteriori* projection approximation criterion described in Section 3.3.3. More concretely, we use the Householder reduction to recursively update the tracked subspace basis $\boldsymbol{Q}(t)$ of dimension $N \times r$ considering the new data point:

$$\begin{bmatrix} \boldsymbol{S}(t) \\ 0 \dots 0 \end{bmatrix} = \boldsymbol{H}(t) \begin{bmatrix} \alpha \boldsymbol{S}(t-1) + \boldsymbol{h}(t)\boldsymbol{h}^\top(t) \\ Z^{1/2}(t)\boldsymbol{h}^\top(t) \end{bmatrix} \tag{4-1}$$

$$\begin{bmatrix} \boldsymbol{Q}(t) & \boldsymbol{z}_q(t) \end{bmatrix} = \begin{bmatrix} \boldsymbol{Q}(t-1) & \bar{\boldsymbol{z}}_\perp(t) \end{bmatrix} \boldsymbol{H}(t) \tag{4-2}$$

where $\bar{\boldsymbol{z}}_\perp(t)$ is the unit-norm version of the complement of the orthogonal projection of $\boldsymbol{z}(t)$ onto the *principal* subspace spanned by the column vectors of $\boldsymbol{Q}(t-1)$ and $Z^{1/2}$ is the norm of this complement. The S-matrix is strongly dominated by its diagonal elements, whose values approximate the principal eigenvalues of the covariance, so we can interpret the transformation as a compressor that keeps most of the energy in the principal eigenvalues given the new input information. This is not an arbitrary heuristic, but follows from the orthogonal iteration principal described in Section 3.3.2.

4.2
Tracking the principal subspace rank

Most subspace tracking algorithms have the dimensionality r of the principal subspace given as a parameter. However, an algorithm that can automatically infer the latent dimensions is widely applicable, especially for monitoring changes in the underlying data stream.

We propose to adapt r, so that we maintain a high percentage f_E of the energy $E(t)$. Energy thresholding is a common method to estimate the number of principal components [Jolliffe, 2002] and is also adopted in SPIRIT [Papadimitriou et al., 2005, Papadimitriou, 2005, Sun, 2007], which has already shown to be useful for detecting unusual patterns in data center measurements [Hoke et al., 2006]. The term energy is common in the signal processing literature and $E(t)$ is defined as

$$E(t) = \sum_{t=1}^{T} \|\boldsymbol{z}(t)\|^2 = \sum_{t=1}^{T} \sum_{i=1}^{N} \boldsymbol{z}_i^2(t). \tag{4-3}$$

We define T as the total number of intervals read so far. We let $\tilde{\boldsymbol{z}}(t)$ be the reconstruction of $\boldsymbol{z}(t)$ based on the previously learnt projection matrix, defined as follows:

$$\tilde{\boldsymbol{z}}(t) = \boldsymbol{Q}(t-1)\boldsymbol{Q}^T(t-1)\boldsymbol{z}(t) = \boldsymbol{Q}(t-1)\boldsymbol{h}(t). \tag{4-4}$$

Similarly, we have the energy $\tilde{E}(t)$ of the reconstruction $\tilde{\boldsymbol{z}}$ defined as

$$\tilde{E}(t) = \sum_{t=1}^{T} \|\tilde{\boldsymbol{z}}(t)\|^2. \tag{4-5}$$

Lemma 3 *Assuming the basis vectors $\boldsymbol{q}_i$, $1 \leq i \leq r$ are orthonormal, we have*

$$\tilde{E}(t) = \frac{1}{T}\sum_{t=1}^{T} \|\boldsymbol{h}(t)\|^2 = \frac{T-1}{T}\tilde{E}(t-1) + \frac{1}{T}\|\boldsymbol{h}(t)\|.$$

Proof: It is straightforward from applying Pythagorean theorem and using the orthonormality of the vectors $\boldsymbol{q}_i$, $1 \leq i \leq r$

$$\begin{aligned}\|\tilde{\boldsymbol{z}}(t)\|^2 &= \|h_1(t)\boldsymbol{q}_1(t-1) + \ldots + h_r(t)\boldsymbol{q}_r(t-1)\|^2 \\ &= h_1^2(t)\|\boldsymbol{q}_1(t-1)\|^2 + \ldots + h_r^2(t)\|\boldsymbol{q}_r(t-1)\|^2 \\ &= h_1^2(t) + \ldots + h_r^2(t) = \|\boldsymbol{h}(t)\|^2\end{aligned}$$

The result follows from summing over T. □

This underlies the importance of a subspace tracker that guarantees orthonormal estimates for the projection matrix. For instance, SPIRIT requires an re-orthonormalization step for Lemma 3 to hold, and this amounts to an extra of $\mathcal{O}\big(Nr^2\big)$ flops[1]. Our algorithm does not need this extra step, and updates the Q-matrix maintaining its orthonormality by construction.

In terms of parameters, we have a low-energy f_E and a high-energy F_E thresholds, and we keep enough number of latent variables r so that the retained energy is within the range $[f_E E(t), F_E E(t)]$. Whenever it gets outside these bounds, we increase or decrease by one unit accordingly. The main steps for the online rank estimation are:

- Estimate the high-dimensional data energy $E(t)$ incrementally from the sum of squares of $\boldsymbol{z}_i$, for all $1 \leq i \leq N$.
- Estimate the energy $\tilde{E}(t)$ of the current $r(t)$ latent variables.
- Adapt $r(t+1)$ if necessary. We add a new basis if the current latent variables capture too little energy, i.e., $\tilde{E}(t) < f_E E(t)$, or drop one dimension if the maintained energy is too high, i.e., $\tilde{E}(t) > F_E E(t)$.

The following lemma proves that the above steps guarantee the relative reconstruction error is within the specified interval $[f_E E(t), F_E E(t)]$.

Lemma 4 *The relative squared error of the reconstruction satisfies*

$$1 - F_E \leq \frac{\sum_{t=1}^{T} \|\boldsymbol{z}(t) - \tilde{\boldsymbol{z}}(t)\|^2}{\sum_{t=1}^{T} \|\boldsymbol{z}(t)\|^2} \leq 1 - f_E$$

[1]This computational complexity is necessary for all orthonormalization procedures such as the classical and stabilized Gram-Schmidt as well as QR decomposition algorithms.

Proof: The measure $\boldsymbol{z}_\perp(t) = \boldsymbol{z}(t) - \tilde{\boldsymbol{z}}(t)$ has been mentioned before in Section 3.3.4 where $\boldsymbol{z}(t) \cdot \boldsymbol{z}_\perp(t) = 0$. Therefore, we have

$$\begin{aligned}\|\boldsymbol{z}(t) - \tilde{\boldsymbol{z}}(t)\|^2 &= \tilde{\boldsymbol{z}}^\top(t)[\boldsymbol{z}(t) - \tilde{\boldsymbol{z}}(t)] - \underbrace{\boldsymbol{z}^\top(t)[\boldsymbol{z}(t) - \tilde{\boldsymbol{z}}(t)]}_{=0 \quad \text{(by orthogonality)}} \\ &= \|\boldsymbol{z}(t)\|^2 - \|\tilde{\boldsymbol{z}}(t)\|^2 \\ &= \|\boldsymbol{z}(t)\|^2 - \|\boldsymbol{h}(t)\|^2 \quad \text{(by Lemma 3).}\end{aligned}$$

The results follows from the summing over T and from the definitions of E (4-3) and $\tilde{E}$ (4-5). □

In Section 5.3.4, we demonstrate that FRAHST's reconstruction error lies in the given range as expected: particularly when setting a range $[f_E, F_E] = [0.96, 0.98]$, our algorithm correctly maintains a relative reconstruction error between 2% and 4% across various datasets.

4.2.1
Exponential forgetting

We can better exploit the temporal nature of the data by using an exponential forgetting factor, which allows us to follow trend drifts over time. We use the same α from (3-1) to properly keep track of the energy, discounting it with the same rate. The update at each step is then:

$$E(t+1) = \frac{t-1}{t}\alpha E(t) + \frac{1}{t}\|\boldsymbol{z}(t)\|^2 \tag{4-6}$$

$$\tilde{E}(t+1) = \frac{t-1}{t}\alpha \tilde{E}(t) + \frac{1}{t}\|\tilde{\boldsymbol{z}}(t)\|^2. \tag{4-7}$$

The value of α doest not affect the computational cost of our method and is therefore more appealing than a sliding window to compute sample statistics, as the latter has buffering requirements.

4.2.2
Adapting data structures

Since r may change over time, our algorithm needs to account for resizing of the two internal matrices Q and S matrices with dimensions $N \times r$ and $r \times r$ respectively. In order to best preserve the properties of each matrix and subsequently minimize losses in the quality of the tracked subspace, we devise two simple heuristics to restructure both matrices. When r is incremented, we append the normalized orthogonal error given by the current updated

projection matrix:

$$\boldsymbol{z}_{\perp}^{\dagger}(t) = \boldsymbol{z}(t) - \boldsymbol{Q}(t)\boldsymbol{Q}^{\top}(t)\boldsymbol{z}(t) \tag{4-8}$$

$$\boldsymbol{Q}(t+1) = \begin{bmatrix}\boldsymbol{Q}(t) & \boldsymbol{z}^{\dagger}(t)/\|\boldsymbol{z}^{\dagger}(t)\|\end{bmatrix} \tag{4-9}$$

where $\boldsymbol{q}_{r+1}$ now servers as an instantaneous estimate for the new basis able to capture interesting information in the new direction. Even though we relaxed the upper-right triangular shape constraint of the S-matrix, in the case of a increase in r we restructure the matrix by adding the following estimate to its new diagonal element:

$$\boldsymbol{S}(t+1) = \begin{bmatrix}\boldsymbol{S}(t) & 0 \\ 0 & \|\boldsymbol{z}_{\perp}^{\dagger}(t)\|^2\end{bmatrix}. \tag{4-10}$$

In the case when r is decremented, the two matrices are simply truncated from $r \times r$ to $(r-1)\times(r-1)$ and we discard the values from the previous entries. The motivation for these updates comes mainly from similar ideas employed in [Yang, 1995b] and while other heuristics were experimented, including very simple ones such as initializing with random orthonormal vectors, they all severely damage the subspace estimates. This completes the necessary steps to our extension which we summarise below in Figure 4.1.

4.3 Exception handling

The zero-input case requires the operation of the algorithm in 'idle mode', which enables the algorithm to handle cases of vanishing inputs. It can be seen in (4-11a) and (4-11b) that a null input will zero both $Z(t)$ and $\boldsymbol{h}(t)$. Hence no updating of the basis estimate in $\boldsymbol{Q}(t)$ is necessary because there is no innovation in a zero input. We add a check to test the condition $Z(t) < \sigma$ before (4-11c) to bypass all remaining computations for that step. The singularity threshold σ is machine and platform dependent but is a positive constant very close to zero.

Fortunately, our rank estimation routine will already guarantee that the S-matrix is not rank deficient, hence we do not need to handle special conditions elsewhere.

4.4 Achieving lower asymptotic computational complexity

Our rank estimation is dominated by $2Nr$ flops from computing the reconstruction error based on the updated projection matrix (4-11n), hence

parameters: $0 < f_E < F_E < 1$ and $0 < \alpha < 1$
$r(0) \leftarrow 1$
$\boldsymbol{Q}(0) \leftarrow$ random orthonormal
$\boldsymbol{S}(0) \leftarrow \sigma \boldsymbol{I}$, where σ is a small positive constant
`for` $t = 1, 2, \ldots$

read $\boldsymbol{z}(t) \in \Re^N$

$$\boldsymbol{h}(t) = \boldsymbol{Q}^\top(t-1)\boldsymbol{z}(t) \tag{4-11a}$$

$$Z(t) = \boldsymbol{z}^\top(t)\boldsymbol{z}(t) - \boldsymbol{h}^\top(t)\boldsymbol{h}(t) \tag{4-11b}$$

$$\boldsymbol{X}(t) = \alpha \boldsymbol{S}(t-1) + \boldsymbol{h}(t)\boldsymbol{h}^\top(t) \tag{4-11c}$$

$$\boldsymbol{X}^\top(t)\boldsymbol{b}(t) = Z^{1/2}\boldsymbol{h}(t) \xrightarrow{\text{solve}} \boldsymbol{b}(t) \tag{4-11d}$$

$$\varphi^2(t) = \frac{1}{2} + \frac{1}{\sqrt{4(\boldsymbol{b}^\top(t)\boldsymbol{b}(t)+1)}} \tag{4-11e}$$

$$\delta(t) = \frac{\varphi(t)}{Z^{1/2}(t)} \tag{4-11f}$$

$$\boldsymbol{v}(t) = \left[\frac{1-2\varphi^2(t)}{2\varphi(t)}\right]\boldsymbol{b}(t) \tag{4-11g}$$

$$\boldsymbol{S}(t) = \boldsymbol{X}(t) - \frac{1}{\delta(t)}\boldsymbol{v}(t)\boldsymbol{h}^\top(t) \tag{4-11h}$$

$$\boldsymbol{e}(t) = \delta(t)\boldsymbol{z}(t) - \boldsymbol{Q}(t-1)\left[\delta(t)\boldsymbol{h}(t) - \boldsymbol{v}(t)\right] \tag{4-11i}$$

$$\boldsymbol{Q}(t) = \boldsymbol{Q}(t-1) - 2\boldsymbol{e}(t)\boldsymbol{v}^\top(t) \tag{4-11j}$$

$$E(t) = \alpha E(t-1) + \|\boldsymbol{z}\|^2 \tag{4-11k}$$

$$\tilde{E}(t) = \alpha \tilde{E}(t-1) + \|\boldsymbol{h}\|^2 \tag{4-11l}$$

`if` $\tilde{E}(t) < f_E E(t)$ `and` $r(t) < N$ (4-11m)

$$\boldsymbol{z}_\perp^\dagger(t) = \boldsymbol{z}(t) - \boldsymbol{Q}(t)\boldsymbol{Q}^\top(t)\boldsymbol{z}(t) \tag{4-11n}$$

$$\boldsymbol{Q}(t+1) = \begin{bmatrix}\boldsymbol{Q}(t) & \boldsymbol{z}^\dagger(t)/\|\boldsymbol{z}^\dagger(t)\|\end{bmatrix} \tag{4-11o}$$

$$\boldsymbol{S}(t+1) = \begin{bmatrix}\boldsymbol{S}(t) & \boldsymbol{0} \\ \boldsymbol{0} & \|\boldsymbol{z}_\perp^\dagger(t)\|^2\end{bmatrix} \tag{4-11p}$$

$$r(t+1) = r(t) + 1 \tag{4-11q}$$

`else if` $\tilde{E}(t) > F_E E(t)$ `and` $r(t) > 1$ (4-11r)

$$\boldsymbol{Q}(t+1) = \text{ } \textit{delete the rightmost column of } \boldsymbol{Q}(t) \tag{4-11s}$$

$$\boldsymbol{S}(t+1) = \text{ } \textit{delete bottom row and rightmost column of } \boldsymbol{S}(t) \tag{4-11t}$$

$$r(t+1) = r(t) - 1 \tag{4-11u}$$

Figure 4.1: FRAHST algorithm

the entire algorithm has time complexity of $5Nr + \mathcal{O}\big(N + r^3\big)$ per update. The space complexity $\mathcal{O}\big(Nr\big)$ is similarly very low.

The value of r depends on the threshold range and the actual data read at each interval t. For example, in the specific datasets we experimented, r

rarely exceeds the value of 6 even for N larger than 200 when capturing 96% of the energy. Despite the fact that $r \ll N$ in most cases, in the worst case analysis we have $r = N$ and one can perceive the typical $\mathcal{O}(r^3)$ complexity for solving the systems of linear equations in step (4-11d) as a stumbling block. Gaussian elimination or LU decomposition are the typical[2] algorithms of choice and require approximately r^3 flops. It is theoretically possible to achieve $\mathcal{O}(r^{2.376})$ by the asymptotically fastest known matrix multiplication (but impractical) Coppersmith–Winograd algorithm [Coppersmith and Winograd, 1990, Robinson, 2005] or with the more realistic Strassen method that requires about $4.7r^{2.807}$ flops [Bailey et al., 1990]. But neither alternatives are optimal. In [Strobach, 2009a], the author suggests that the recursivity that is inherent in the subspace tracking algorithm can be exploited by recursive updating factorizations of inner matrices, but not much detail is provided. We follow this thread and show how to apply the same suggested ideas for our rank-adaptive extension and reduce the complexity of step (4-11d) to $\mathcal{O}(r^2)$.

4.4.1 Efficiently updating QR-factorizations

Instead of refactoring the X matrix on every update from scratch, it is much more efficient to update each factor as necessary and a procedure for such recurrent updates is widely known and require $\mathcal{O}(r^2)$ flops (Chapter 12.5 in Golub and Van Loan [1996] and Gill et al. [1972]). The idea is to work with the QR decompositions of the X- and S- $r \times r$ square matrices and recognize that (4-11c) and (4-11h) are 'QR = QR + rank one' updates that can be expressed as

$$\boldsymbol{X}(t) = \boldsymbol{X}^Q(t)\boldsymbol{X}^R(t) = \boldsymbol{S}^Q(t)\boldsymbol{S}^R(t) - \boldsymbol{h}^\top(t)\boldsymbol{h}(t) \tag{4-12}$$

$$\boldsymbol{S}(t) = \boldsymbol{S}^Q(t)\boldsymbol{S}^R(t) = \boldsymbol{X}^Q(t)\boldsymbol{X}^R(t) - 1/\delta(t)\boldsymbol{v}(t)\boldsymbol{h}^\top(t) \tag{4-13}$$

We illustrate the procedure considering the case $\boldsymbol{A}^Q\boldsymbol{A}^R = \boldsymbol{A} \in R^{r\times r}$, where we need to compute the QR factorization of $\boldsymbol{A} + \boldsymbol{b}\boldsymbol{c}^\top = \boldsymbol{A}_1^Q\boldsymbol{A}_1^R$ and $\boldsymbol{b}, \boldsymbol{c} \in R^r$. The expression is written in the form $\boldsymbol{A} + \boldsymbol{b}\boldsymbol{c}^\top = \boldsymbol{A}^Q(\boldsymbol{A}^R + \boldsymbol{w}\boldsymbol{c}^\top)$ where $\boldsymbol{w} = \boldsymbol{A}^Q\boldsymbol{b}$. We use the procedure detailed in [Golub and Van Loan, 1996] to restore the QR factorization of $\boldsymbol{A}$ which relies on computing two sequences

[2]Most software packages seem to call the DGESV from LAPACK to compute the solution of linear equations.

J and G of Givens rotations as shown below:

$$\texttt{QR-UPDATE}(\boldsymbol{A}^Q, \boldsymbol{A}^R, \boldsymbol{b}, \boldsymbol{c}) \tag{4-14}$$

$$\boldsymbol{J}_1^\top \dots \boldsymbol{J}_{m-1}\boldsymbol{w} = \pm\|\boldsymbol{w}\|\boldsymbol{e}_1 \tag{4-15}$$

$$\boldsymbol{J}_1^\top \dots \boldsymbol{J}_{m-1}(\boldsymbol{A}^R + \boldsymbol{w}\boldsymbol{c}^\top) = \boldsymbol{H} \pm \|\boldsymbol{w}\|\boldsymbol{e}_1\boldsymbol{b}^\top = \boldsymbol{H}_1 \tag{4-16}$$

$$\boldsymbol{G}_{m-1}^\top \dots \boldsymbol{G}_1^\top \boldsymbol{H}_1 = \boldsymbol{A}_1^R \tag{4-17}$$

$$\boldsymbol{A}_1^Q = \boldsymbol{A}^Q \boldsymbol{J}_{m-1} \dots \boldsymbol{J}_1 \boldsymbol{G}_1 \dots \boldsymbol{G}_{m-1} \tag{4-18}$$

$$\textit{output } \boldsymbol{A}_1^Q, \boldsymbol{A}_1^R \tag{4-19}$$

where $\boldsymbol{e}_1 \equiv [1\ 0\ \dots\ 0]^\top \in R^r$. Each $\boldsymbol{J}_k$ is a Given rotation in planes k and $k+1$ [Algorithm 5.1.3 in Golub and Van Loan, 1996]. $\boldsymbol{H_1}$ can be shown to be an 'almost upper triangular' Hessenberg matrix and its QR factorization (4-17) can be computed by applying the G rotations in $\mathcal{O}(r^2)$ [Algorithm 5.2.3 in Golub and Van Loan, 1996]. The entire update requires about $26r^2$ flops. Once we have the QR factors, it is straight-forward to solve the linear system from (4-11d) with complexity $\mathcal{O}(r^2)$ by using back-substitution, which follows from pre-multiplying the transpose of $\boldsymbol{X}^Q$ and from the fact that $\boldsymbol{X}^R$ is an upper triangular matrix (step (4-20b)).

We are now left with the task to account for the resizing of the QR factors of the S-matrix. Similarly, we use four procedures based on Givens rotations from [Golub and Van Loan, 1996, Hajek, 2009] for both appending and deleting column and row vector of the original matrix according to steps (4-11p) and (4-11t) whilst maintaining the QR structure with quadratic complexity. We do not give all details here, but all the routines `QR-INSERT-ROW`, `QR-INSERT-COL`, `QR-DELETE-COL` and `QR-DELETE-ROW` are all very much alike to the above `QR-UPDATE` and surgically manipulate the values of the upper triangular A^R-matrix via Givens rotations which is then used to produce a Hessenberg matrix to yield the final two rotations that are used to update the A^Q-matrix. We give the pseudo-code for the asymptotically faster modification of FRAHST in Figure 4.2.

Considering that each routine costs roughly a total of $26r^2$ flops, and adding r^2 operations from the back-substitution step totals about $5Nr + 157r^2$ which suggests that the asymptotically faster modified version only pays off when $r > 157$. In practice, any implementation can switch to this version when the dimension of the principal subspace is that high while using few operations for the most common lower rank case. Regarding the implementation, we note that MATLAB, for instance, only offers the `QR-UPDATE` function[3] but all

[3]See the documentation for the *qrupdate* at `http://www.mathworks.com/`.

Initialize $\boldsymbol{S}^Q(0) = \boldsymbol{S}^R(0) = \sigma \boldsymbol{I}$
Same steps (4-11a)-(4-11u) except for the following.

Replace (4-11c) and (4-11d) with:

$$\boldsymbol{X}^Q(t)\boldsymbol{X}^R(t) \leftarrow \texttt{QR-UPDATE}(\ \alpha \boldsymbol{S}^Q(t),\ \alpha \boldsymbol{S}^R(t),\ \boldsymbol{h},\ \boldsymbol{h}\ ) \tag{4-20a}$$

$$\boldsymbol{X}^R\boldsymbol{b} = (\boldsymbol{X}^Q)^\top Z(t)^{1/2}\boldsymbol{h}(t) \xrightarrow{\text{back-substitution}} \boldsymbol{b}(t). \tag{4-20b}$$

Replace (4-11h) with:

$$\boldsymbol{S}^Q(t)\boldsymbol{S}^R(t) \leftarrow \texttt{QR-UPDATE}(\ \boldsymbol{X}^Q(t), \boldsymbol{X}^R(t), 1/\delta(t)\boldsymbol{v}, \boldsymbol{h}\ ). \tag{4-20c}$$

Replace (4-11p) with:

$$\boldsymbol{S}^Q(t+1)\boldsymbol{S}^R(t+1) \leftarrow \texttt{QR-INSERT-ROW}(\ \boldsymbol{S}^Q(t),\ \boldsymbol{S}^R(t)\ ,\ r(t+1),\ \boldsymbol{0}\ ) \tag{4-20d}$$

$$\boldsymbol{S}^Q(t+1)\boldsymbol{S}^R(t+1) \leftarrow \texttt{QR-INSERT-COL}(\ \boldsymbol{S}^Q(t+1),\ \boldsymbol{S}^R(t+1),\ r(t+1),\ [0\ 0\ \ldots\ \|\boldsymbol{z}_\perp^\dagger(t)\|^2]\ )\ . \tag{4-20e}$$

Replace (4-11t) with:

$$\boldsymbol{S}^Q(t+1)\boldsymbol{S}^R(t+1) \leftarrow \texttt{QR-DELETE-COL}(\ \boldsymbol{S}^Q(t),\ \boldsymbol{S}^R(t),\ r(t)\ ) \tag{4-20f}$$

$$\boldsymbol{S}^Q(t+1)\boldsymbol{S}^R(t+1) \leftarrow \texttt{QR-DELETE-ROW}(\ \boldsymbol{S}^Q(t+1),\ \boldsymbol{S}^R(t+1),\ r(t)\ )\ . \tag{4-20g}$$

Figure 4.2: Asymptotically faster FRAHST algorithm with recurrent QR updates.

routines are found in the Fortran package from Hajek [2009] which is greatly optimized.

4.4.2 Efficiently updating LU-factorizations

We explore a second alternative to reduce the $\mathcal{O}(r^3)$ term from the original complexity in terms of fast Bennett LU-factor updating [Bennett, 1965], cited in [Strobach, 2009a] and the algorithm is detailed in [Stange et al., 2007] and it only requires $4r^2$ flops for updating. Similarly to the previous extension, this allows the linear system of equations to be solved in $\mathcal{O}(r^2)$ by exploiting the 'LU = LU + rank one' updates in each step. However, now we are dealing with the lower and upper triangular factors of the S- and X-matrices. Analogous to the previous section, the main goal is to update the LU decomposition after a rank-one update:

$$\underbrace{\boldsymbol{A}^L\boldsymbol{A}^U}_{\boldsymbol{A}} + \boldsymbol{b}\boldsymbol{c}^\top = \underbrace{\boldsymbol{A}_1^Q\boldsymbol{A}_1^R}_{\text{New LU factors}} \tag{4-21}$$

which can be accomplished with an algorithm such as Bennett's algo-

rithm, which is listed below [from Stange et al., 2007]:

$$\begin{aligned}
&\texttt{LU-UPDATE}(\boldsymbol{A}^L, \boldsymbol{A}^U, \boldsymbol{b}, \boldsymbol{c}) \\
&\texttt{for } 1 \leq i \leq r && (4\text{-}22)\\
&\quad \boldsymbol{A}^U_{ii} = \boldsymbol{A}^U_{ii} + b_i c_i && (4\text{-}23)\\
&\quad c_i = c_i / \boldsymbol{A}^U_{ii} && (4\text{-}24)\\
&\quad \texttt{for } i+1 \leq j \leq r && (4\text{-}25)\\
&\quad\quad b_j = b_j - b_i \boldsymbol{L}_{ji} && (4\text{-}26)\\
&\quad\quad \boldsymbol{L}_{ji} = \boldsymbol{L}_{ji} + c_i b_j && (4\text{-}27)\\
&\quad\quad \boldsymbol{U}_{ij} = \boldsymbol{U}_{ij} + b_i c_j && (4\text{-}28)\\
&\quad\quad c_j = c_j - c_i \boldsymbol{U}_{ij} && (4\text{-}29)
\end{aligned}$$

Once we have the LU factors for the X- matrix, solving the linear system of equations from (4-11d) is a direct application of both forward- and back-substitutions. The pseudo-code for the asymptotically faster modification of FRAHST is given in Figure 4.3.

Initialize $\boldsymbol{S}^L(0) = \boldsymbol{S}^U(0) = \sigma \boldsymbol{I}$

Same steps (4-11a)-(4-11u) except for the following.

Replace (4-11c) and (4-11d) with:

$$\boldsymbol{X}^L(t)\boldsymbol{X}^U(t) \leftarrow \texttt{LU-UPDATE}(\ \alpha \boldsymbol{S}^L(t),\ \alpha \boldsymbol{S}^U(t),\ \boldsymbol{h},\ \boldsymbol{h}\ ) \quad (4\text{-}30a)$$

$$\boldsymbol{X}^L \boldsymbol{y}(t) = Z(t)^{1/2} \boldsymbol{h}(t) \xrightarrow{\text{forward-substitution}} \boldsymbol{y}(t). \quad (4\text{-}30b)$$

$$\boldsymbol{X}^U \boldsymbol{b} = \boldsymbol{y}(t) \xrightarrow{\text{back-substitution}} \boldsymbol{b}(t). \quad (4\text{-}30c)$$

Replace (4-11h) with:

$$\boldsymbol{S}^L(t)\boldsymbol{S}^U(t) \leftarrow \texttt{LU-UPDATE}(\ \boldsymbol{X}^L(t), \boldsymbol{X}^U(t), 1/\delta(t)\boldsymbol{v}, \boldsymbol{h}\ ). \quad (4\text{-}30d)$$

Replace (4-11p) with:

$$\boldsymbol{S}^L(t+1) = \begin{bmatrix} \boldsymbol{S}^L(t) & \boldsymbol{0} \\ \boldsymbol{0} & \|\boldsymbol{z}^\dagger_\perp(t)\| \end{bmatrix} \quad \boldsymbol{S}^U(t+1) = \begin{bmatrix} \boldsymbol{S}^U(t) & \boldsymbol{0} \\ \boldsymbol{0} & \|\boldsymbol{z}^\dagger_\perp(t)\| \end{bmatrix} \quad (4\text{-}30e)$$

Replace (4-11t) with:

$$\boldsymbol{S}^L(t+1) = \textit{delete bottom row and right most column of } \boldsymbol{S}^L(t) \quad (4\text{-}30f)$$

$$\boldsymbol{S}^U(t+1) = \textit{delete bottom row and right most column of } \boldsymbol{S}^U(t) \quad (4\text{-}30g)$$

Figure 4.3: Asymptotically faster FRAHST algorithm with recurrent LU updates.

The total complexity is about $5Nr + 10r^2$ due to two $4r^2$ LU updates and two r^2 substitutions for solving the linear system. This is the lowest complexity

we achieve for the FRAHST algorithm. However, Bennett's algorithm is known to cause numerical stability problems during the updating procedure [Stange et al., 2007], and we indeed experience a loss of quality in the subspace estimates for this faster version in practice. Therefore, in general we are recommending the previous QR version for a more stable approach that performs equally well as the original. In Chapter 5, we compare the performance of all the proposed versions (see Figure 5.4(c)).

Summary of the algorithm

The latent variables $\boldsymbol{h}(t)$ give us a compact representation of the raw data $\boldsymbol{z}(t)$ and FRAHST guarantees high reconstruction accuracy (in terms of the relative squared error, which is less than $1 - f_E$). When our streams are highly correlated, as we often expect to be the case, the latent dimension r is much smaller than the number N of streams and the Q-matrix captures the linear combinations between them. Traditional batch methods for estimating the principal components require time that depends on the duration T, and most subspace trackers do not adapt the rank of the principal subspace in an online manner. Our algorithm does not need to store any past values and has *very* low computational requirements and can be applied to several important tasks, such as anomaly detection by monitoring the current tracked rank and forecasting by fitting auto-regressive models on the latent variables for the next interval, especially to handle missing data – the latter techniques were advocated in [Papadimitriou et al., 2005], but FRAHST has a lower dominant complexity $\mathcal{O}(Nr)$ instead of $\mathcal{O}(Nr^2)$ needed for SPIRIT to guarantee the correct expected reconstruction error. This makes our algorithm more attractive for the streaming scenario.

4.5 Real-time Anomaly Detection System

Now, we formalize the anomaly detection procedure based on the FRAHST algorithm, and elaborate on the architecture for implementing a real-time monitoring system.

4.5.1 Raising alarms

Our anomaly detection procedure follows directly from the rank-adaptive nature of FRAHST. We propose to raise alarms whenever there is an increase in rank, as laid out below.

```
last ← 0
for t = 1, 2, ...
  FRAHST(t)                                              (4-31)
  if  r(t) > r(t − 1)                                    (4-32)
    if  t > last + 1:  raise an alarm at t               (4-33)
    last ← t                                             (4-34)
```

Figure 4.4: Anomaly detection routine

The control variable `last` is used to suppress alarms when there are rank increments in consecutive interval, which are likely to be false alarms. The intuition is that a change in the underlying streams might be so great, that more than one additional latent variables are necessary to achieve the same reconstruction error.

4.5.2 Event-driven Architecture

We devise a loose-coupled event-driven architecture where all modules are *publishers* or *subscribers* to an event broker, also known as event/message bus [and widely popularized by Hohpe and Woolf, 2003]. The most important components in a data center can be monitored using the Simple Network Management Protocol (SNMP), which include routers, access servers, switches, bridges, hubs, temperature sensors and computer hosts. The variables accessible via SNMP are organized in hierarchical Management Information Bases (MIBs) and a manager can get information from agents using the UDP protocol, typically via port 161. Not all data is available via SNMP, such as statistics from a running MySql database or an Apache web server. As illustrated in Figure 4.5, specific plugin modules are implemented to obtain all desired information which is then normalized into a JSON[4] format which is sent to the bus to be consumed by different processes via remote calls. We also integrate with Nagios[5] in order to reuse the existing plugins to enable a wide variety of data to be available into our system. This normalized message contains: timestamp, event type identifier and the collected variables along with their names. The user configures, in runtime via a web interface, a model that represents the entities in the monitored infrastructure and are mapped to this raw data accordingly. There are many challenges to implement efficient communication

[4]Javascript Object Notation
[5]A popular opensource monitoring system.

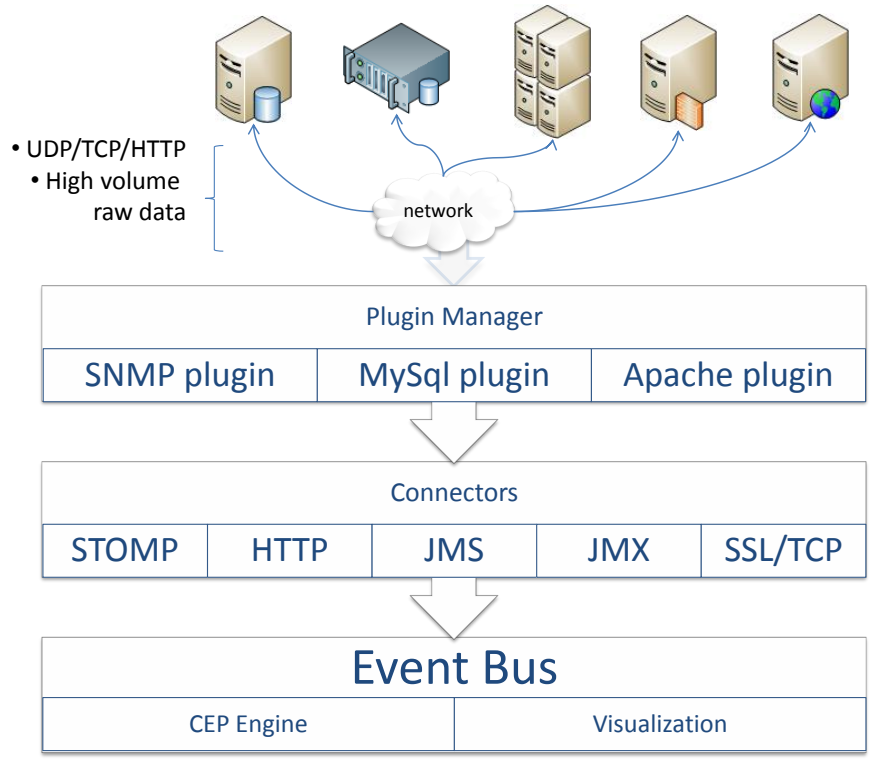

Figure 4.5: Raw asynchronous events are collected and normalized messages are sent to the event bus.

strategies to ensure minimum latency. For this reason most communication is done over a persistent TCP socket using the STOMP[6] protocol.

Different implementations were considered, but we chose Esper[7] as the event stream processing engine[8] to power our solution. It is an excellent open source project, easy to embedded and with the probably most expressive event processing query language available. A CEP engine is analogous to a 'inverted' traditional database, where instead of having queries being made over persistent tuples of data we now have persistent queries that run continuously over streaming data.

The idea is to monitor streams as defined by continuous queries over the incoming raw data. More specifically, the data center operator chooses a set of streams to be monitored together and a query can be placed to join the corresponding underlying streams in order to produce the input vector $\boldsymbol{z}(t)$ for our algorithm at periodic equidistant intervals – the join operator and the output rate of the query are features of the core stream processing engine. This is necessary due to the asynchronous nature of the original events. Figure

[6]Streaming Text Orientated Message Protocol
[7]More information at http://www.espertech.com/.
[8]The terms event stream processing (ESP), complex event processing (CEP) and data stream management system (DSMS) may be used interchangeably.

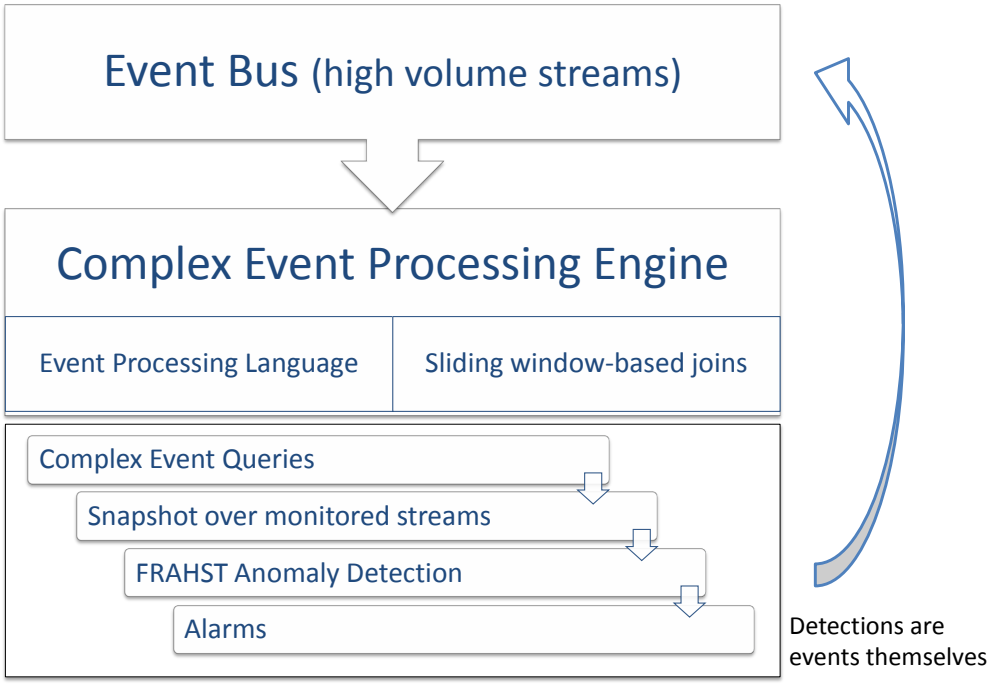

Figure 4.6: We illustrate the processing flow in the system: a query is used to join raw streams into a derived complex event that feeds our algorithm, which is then apt to detect anomalies in variables under surveillance.

4.6 overviews the flow of the anomaly detection system and it is important to realize that we have multiple instances of FRAHST running attached to different queries with different set of streams. For example, one instance will be detecting anomalies from a query that represents data regarding cpu and memory usage from a cluster A, while another will monitor cluster B. This solution is flexible and allows an expert user to leverage his previous knowledge while maintaining an interpretation over the projected latent variables.

We use the adapter pattern for the actual integration with FRAHST where a processor is attached to a continuous query and updates the algorithm upon new events. The step (4-33) in Figure 4.4 notifies a listener which consequently publishes an alarm event to the message broker so further actions are taken; such as update charts, send SMS or XMPP messages. It is interesting to observe that generated alarms can also be considered event streams, and hence can be monitored. The sequence diagram in Figure 4.7 better illustrates the interaction between the different subcomponents in the monitoring system. During the each step interval, data is collected from different sources and is aggregated by user-defined queries. Each query has an associated query listener, which is responsible for forming the input data vector and delegating

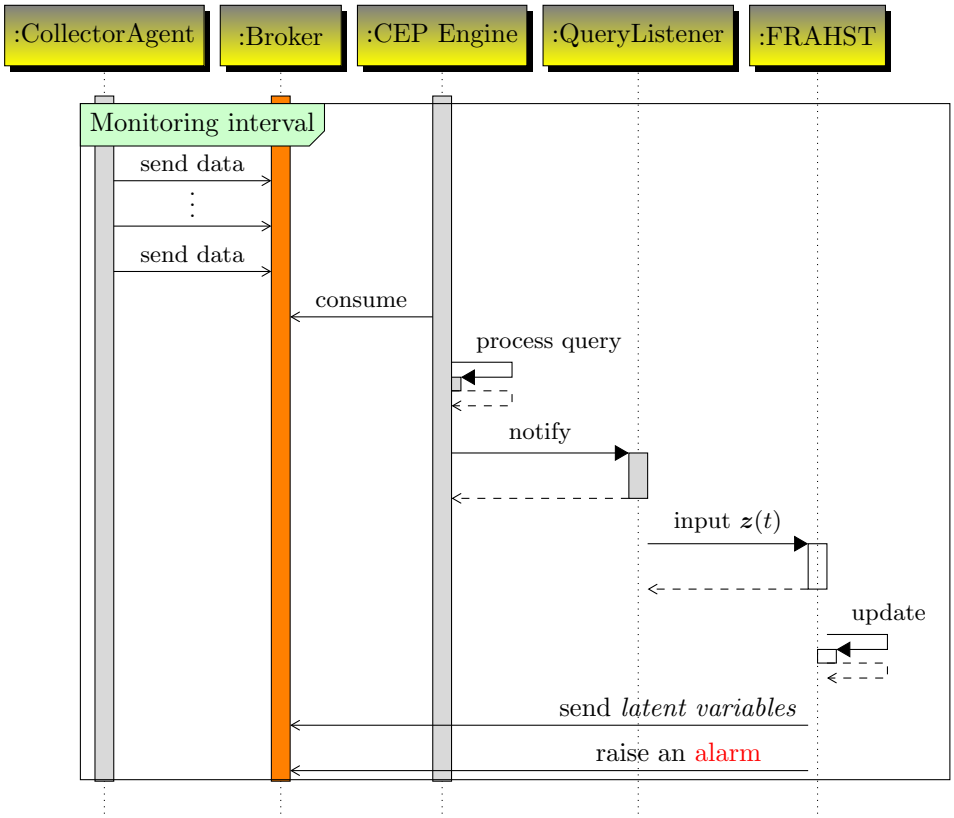

Figure 4.7: Simplified sequence diagram of the update step.

over an adapter to the FRAHST implementation. Although we do not depict it here, a user interface can consume both latent variables and alarm information from the broker for visualization purposes. Most of the system is implemented in the Java language, and ActiveMQ[9] was used for the event broker.

Our solution has been adopted for a real data center, where it has been shown to achieve high throughput and low latency. This use case has been presented at an international forum recently [Clemente and Vieira, 2009].

[9]See http://activemq.apache.org/.