High Performance Computing for Science and Engineering I.

Exercise Set 3

Florian Mahlknecht

2019-11-15

1 Prinicpal Component Analysis

1.1 The Covariance Method

For calculating the covariant matrix a more general approach using the Intel's BLAS implementation delivered in MKL was used.

```
320 void utils::constructCovariance(double *C, const double * const data_T,
321                                   const int N, const int D)
322 {
323   // Construct the covariance matrix (DxD) of the data.
324   // data(n,d)=data_T[d*N+n]
325   // For the covariance follow the row major notation
326   // C(j,k)=C[j*D+k]
327
328   // note that the output is column major,
329   // but it does not matter since it will be symmetric by design
330   double alpha = 1.0 / (N-1);
331
332   cblas_dgemm(CBLAS_LAYOUT::CblasColMajor,
333               CBLAS_TRANSPOSE::CblasTrans,
334               CBLAS_TRANSPOSE::CblasNoTrans,
335               D, D, N, // m n k
336               alpha, data_T, N, data_T, N,
337               0.0, C, D);
338 }
```

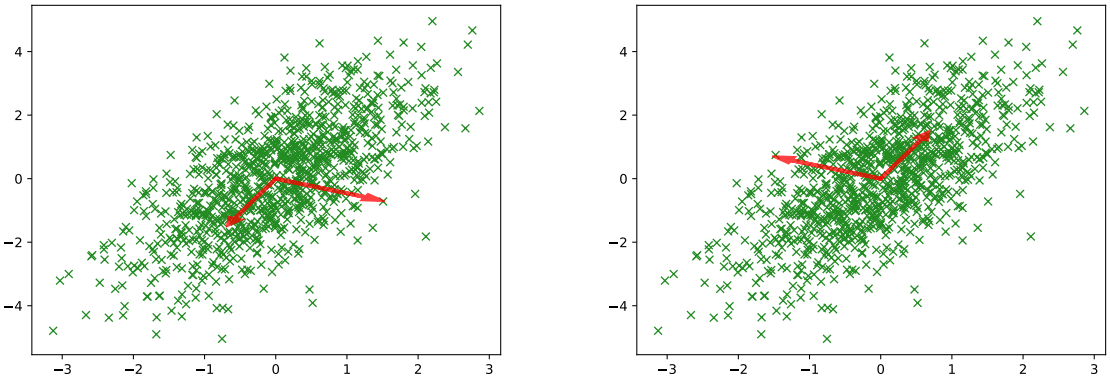
Listing 1: Covariance creation implementation

Listing 1 exploits the transpose flags of BLAS. This serves as an example snippet, for the complete implementation please refer to the attached source code.

1.1.1 2D Results

As fig. 1 shows, the result has been obtained as desired: just the sign changed, which – concerning the direction – does not alter the result.

fmahlknecht@student.ethz.ch
19-945-351



(a) Reference result (Python) (b) Obtained result

Figure 1: Results comparison (expected vs. obtained)

The two computed eigenvectors are numerically:

$$v_1 = \begin{pmatrix} 0.418679 \\ 0.908134 \end{pmatrix} \qquad v_2 = \begin{pmatrix} -0.908134 \\ 0.418679 \end{pmatrix}$$

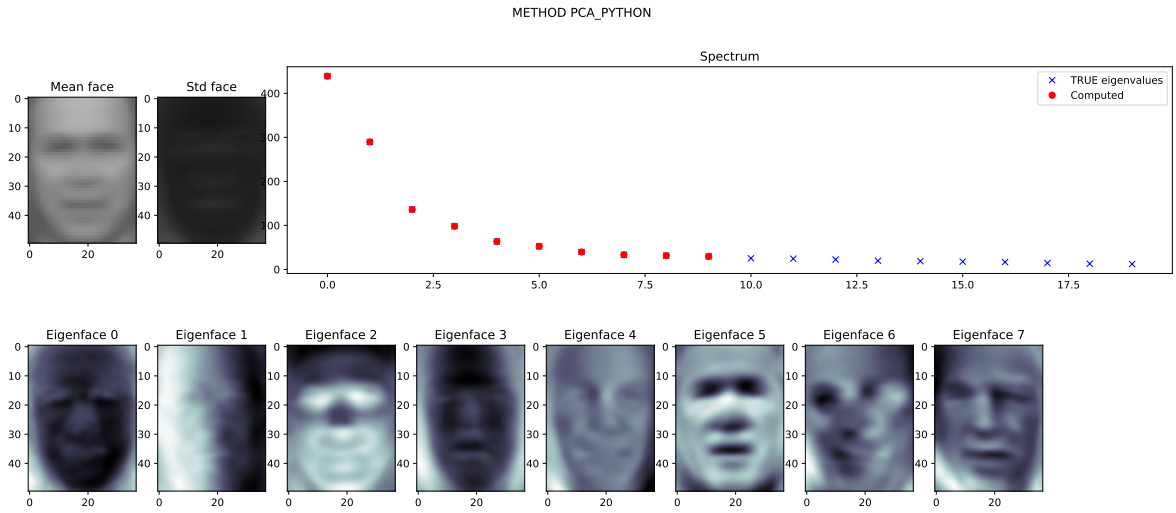
And their respective eigen values:

$$\lambda_1 = 3.45358 \qquad \lambda_2 = 0.438344$$

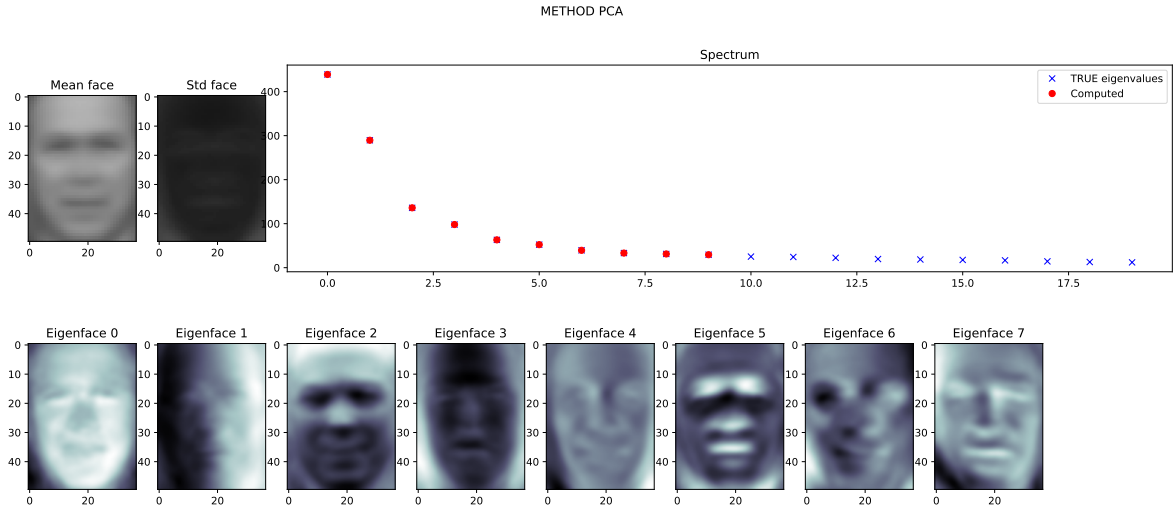
Florian Mahlknecht
fmahlknecht@student.ethz.ch
19-945-351

1.1.2 Face dataset results

The face dataset examples yield similar results, as shown in fig. 2. The inverted sign is hereby reflected in the inverted brightness scale.



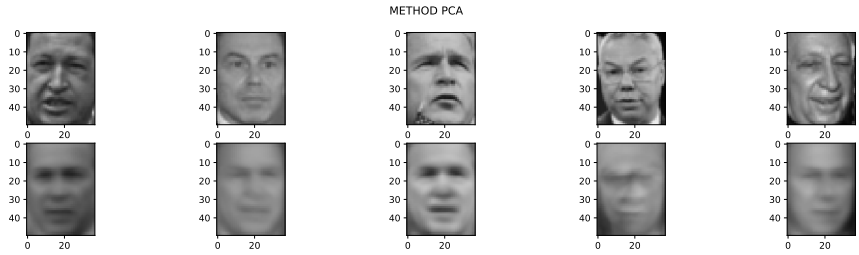
(a) Reference result



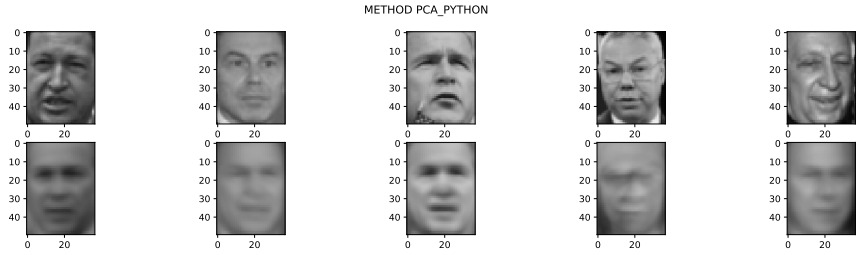
(b) Obtained result

Figure 2: Results comparison (expected vs. obtained)

The reconstruction, yields identical results, given in fig. 3. The reconstruction takes care of the right signs as expected, since the coordinates of the subspace will change signs according to their chosen eigenvectors basis.



(a) Reference result

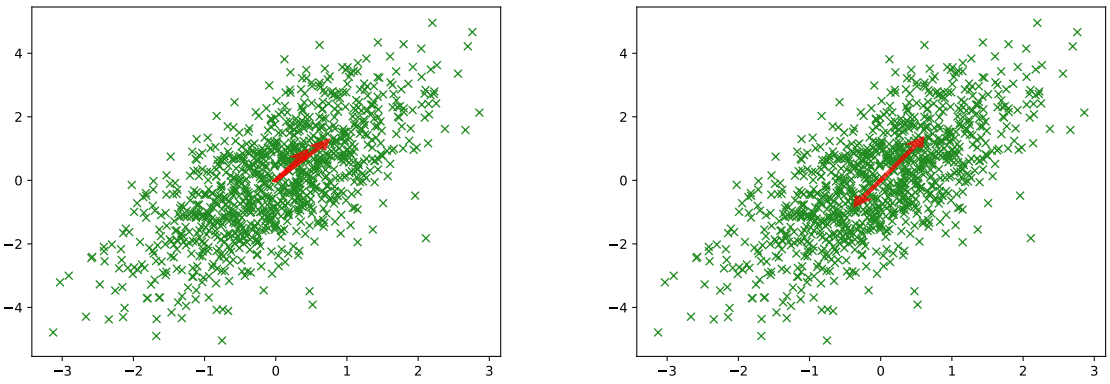


(b) Obtained result

Figure 3: Results comparison (expected vs. obtained)

1.2 Oja's Rule

Applying Oja's rule, we obtain the results shown in fig. 4.



(a) Without gradient normalization (b) With gradient normalization

Figure 4: Obtained eigenvectors using Oja's rule with random initializations

As expected, applying the same update rule, we just get the principal components. They may differ in scaling, but their eigenvalue is always the largest one:

$$\lambda_1 = 3.42569 \qquad \lambda_2 = 3.42559,$$

with respectively

$$v_1 = \begin{pmatrix} 1.00239 \\ 1.74129 \end{pmatrix} \qquad v_2 = \begin{pmatrix} 0.728473 \\ 1.26493 \end{pmatrix}$$

Florian Mahlknecht
fmahlknecht@student.ethz.ch
19-945-351

1.3 Sanger's Rule

The implementation of the Sanger's rule is carried out in the following way:

```
87  void Perceptron::sangersRuleGradient(const double * const input, const int batch_size) {
88    // input [batch_size, nInputs]
89    // output [batch_size, nOutputs]
90    // weights [nInputs, nOutputs]
91
92    forward(input, batch_size);
93    memset(gradient, 0.0, sizeof(double) * nOutputs * nInputs);
94
95    for (int k = 0; k < batch_size; ++k) {
96      for (int i = 0; i < nInputs; ++i) {
97        double sum_weights = 0.0;
98        for (int o = 0; o < nOutputs; ++o) {
99          sum_weights += weights[i*nOutputs + o] * output[k*nOutputs + o];
100
101         gradient[o + i * nOutputs] += output[o + k * nOutputs] *
102               (input[i + k * nInputs] - sum_weights);
103       }
104     }
105   }
106
107   for (int i = 0; i < nInputs * nOutputs; ++i) {
108     gradient[i] = gradient[i] / batch_size;
109   }
110 }
111
```

Listing 2: Sanger's Rule Implementation

1.3.1 2D Dataset

In the 2D case we choose the following parameters:

batch size 32

iterations 1045000

learning rate 1e−7

We get the result shown in fig. 5.

fmahlknecht@student.ethz.ch
19-945-351

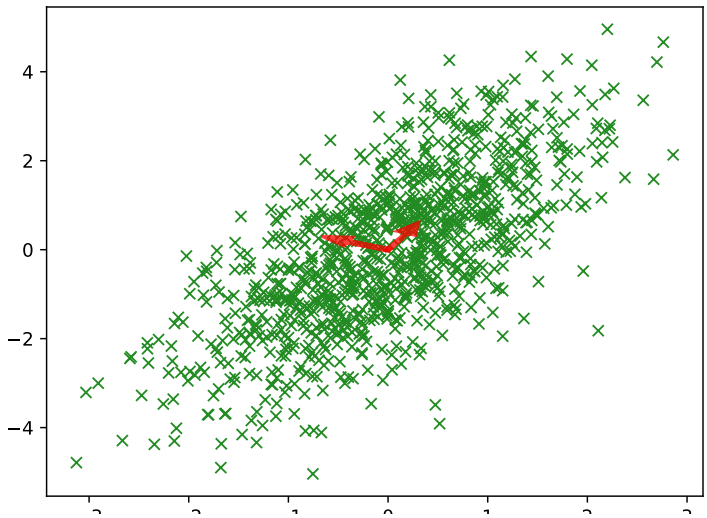


Figure 5: Sanger's 2D component results

Numerically we have:

$$v_1 = \begin{pmatrix} 0.141895 \\ 0.278775 \end{pmatrix} \qquad v_2 = \begin{pmatrix} -0.297604 \\ 0.131818 \end{pmatrix}$$
$$\lambda_1 = 3.44567 \qquad \lambda_2 = 0.438596$$

Which is very similar to the results obtained with the deterministic approach using singular value decomposition, described in section 1.1. It is important to note, however, that the norm of the vectors is not equal to 1 anymore.

By extending the program with plotting the eigenvalues regularly during the iterations we obtain the graph shown in fig. 6.

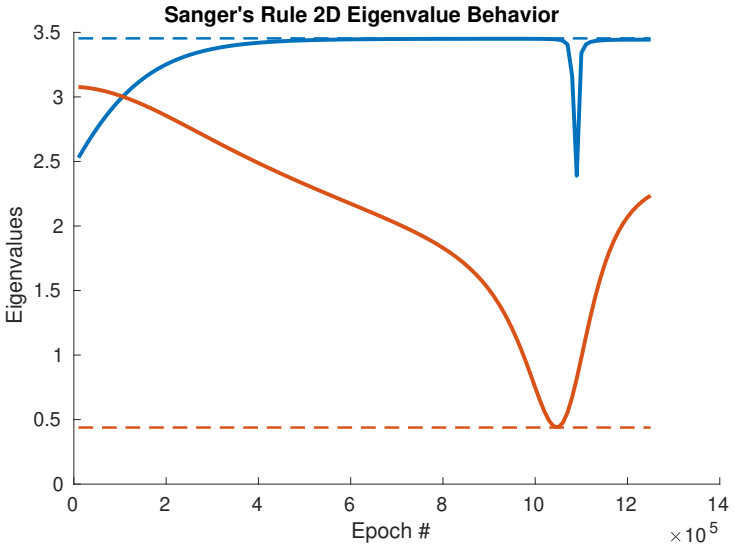


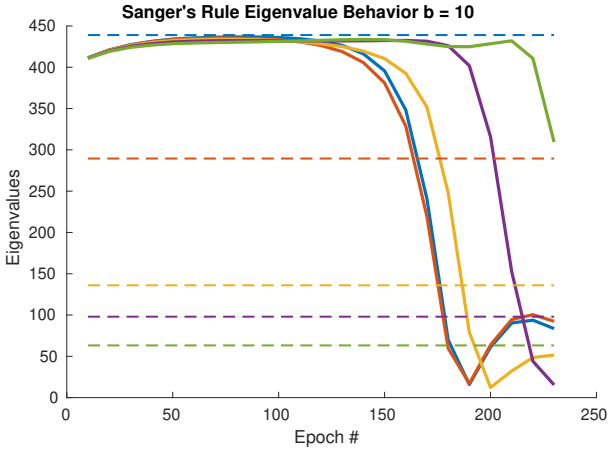
Figure 6: Sanger's 2D eigenvalue behavior

Only very close to the chosen number of iterations, the eigenvalues are actually reaching the true eigenvalues shown as a dashed line.

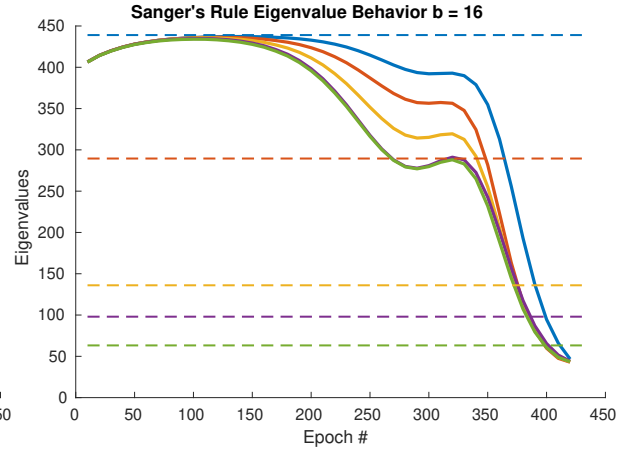
Florian Mahlknecht
fmahlknecht@student.ethz.ch
19-945-351

1.3.2 Faces Dataset

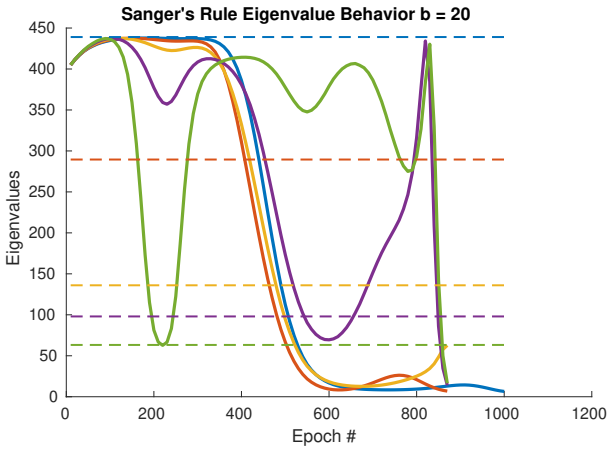
The convergence plot is now used for the first 5 eigenvalues in the faces dataset, in fig. 7 compared with different batch sizes. The learning rate is set to 1e−6, and weights are initialized all equally.



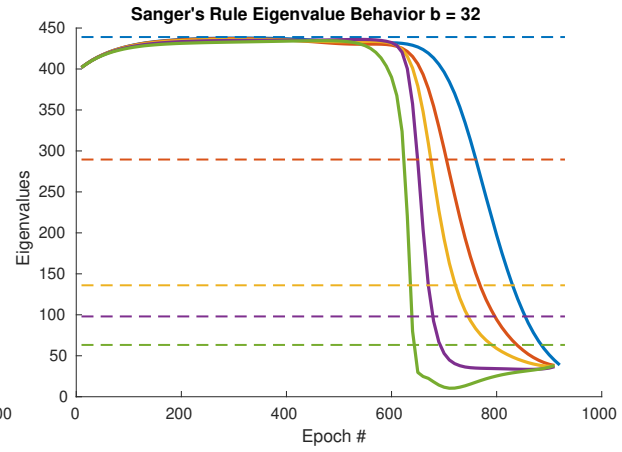
(a) Batch size 10



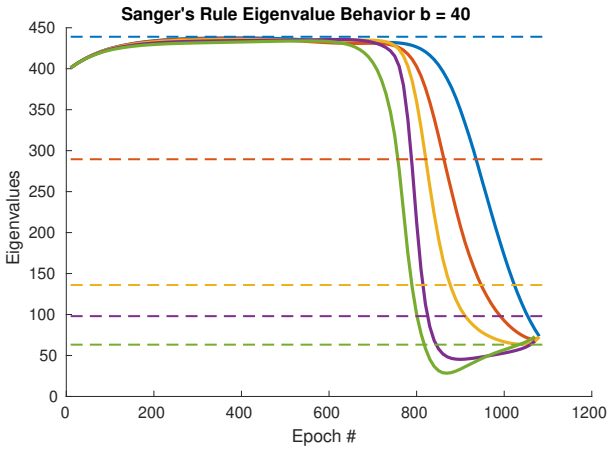
(b) Batch size 16



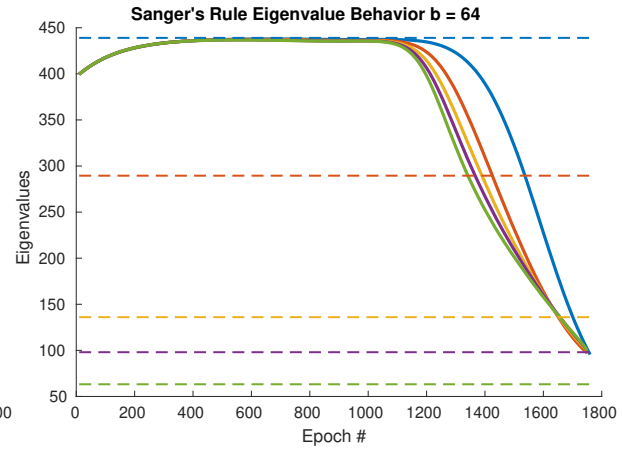
(c) Batch size 20



(d) Batch size 32



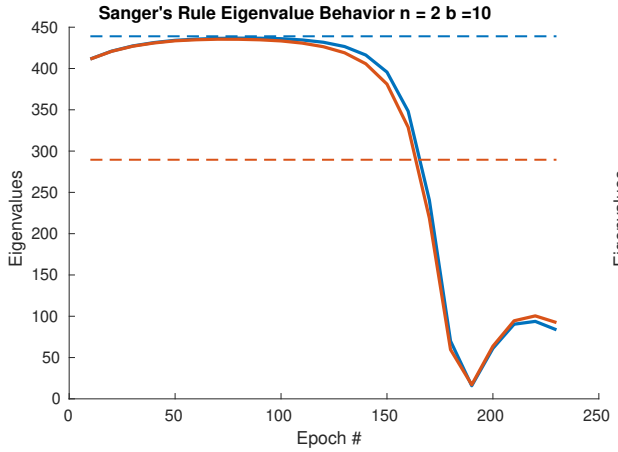
(e) Batch size 40



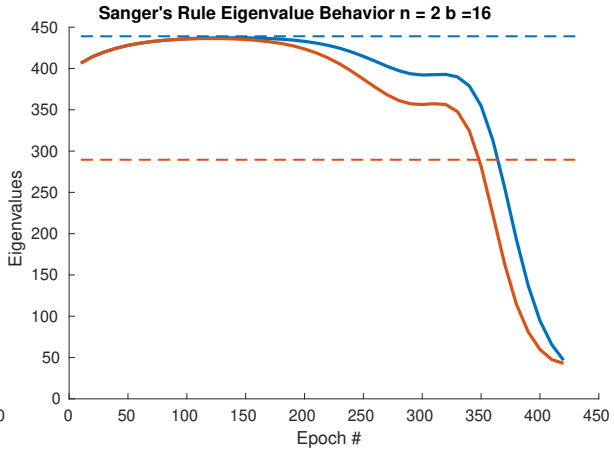
(f) Batch size 64

Figure 7: Sanger's rule convergence behavior

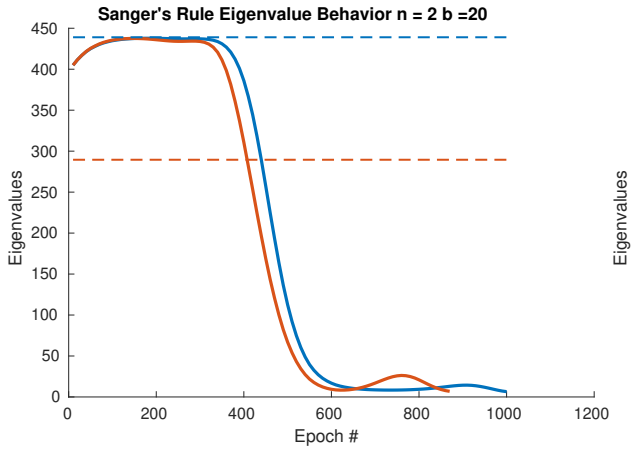
This can be repeated for just the first two components, in fig. 8.



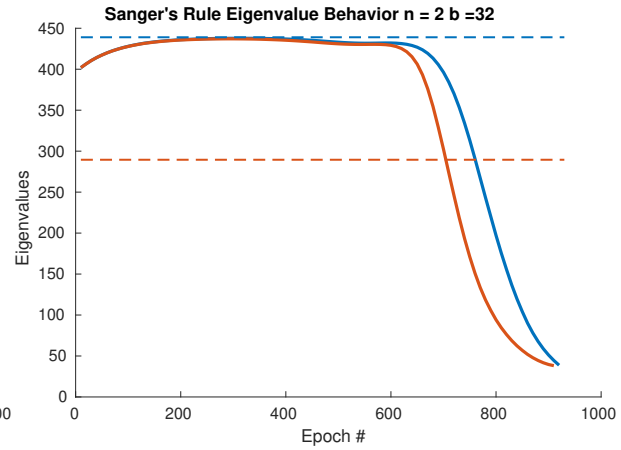
(a) Batch size 10



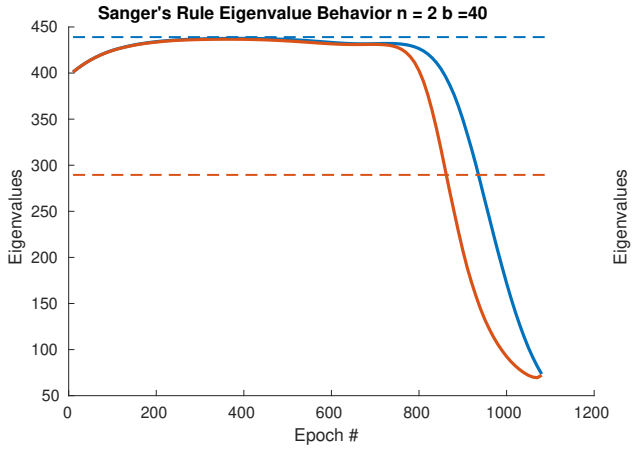
(b) Batch size 16



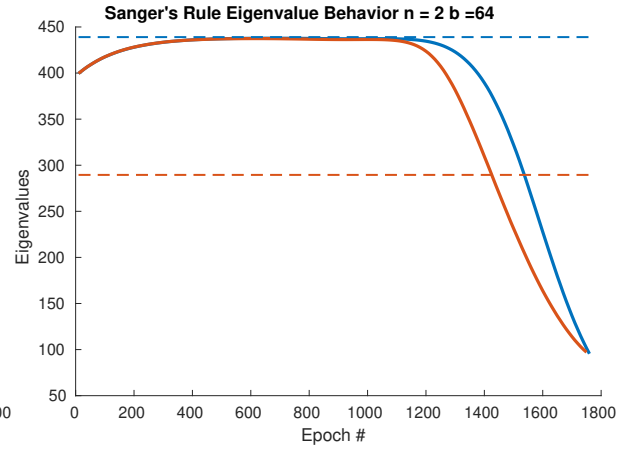
(c) Batch size 20



(d) Batch size 32



(e) Batch size 40

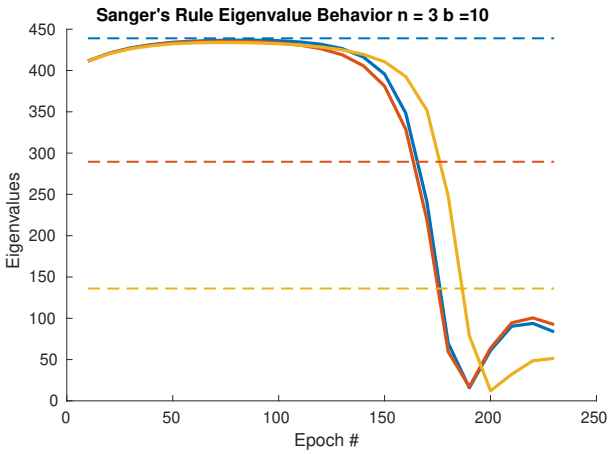


(f) Batch size 64

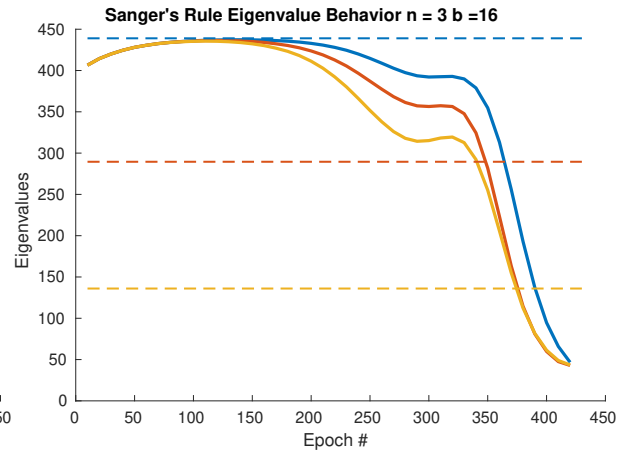
Figure 8: Sanger's rule convergence behavior for 2 components

Florian Mahlknecht
fmahlknecht@student.ethz.ch
19-945-351

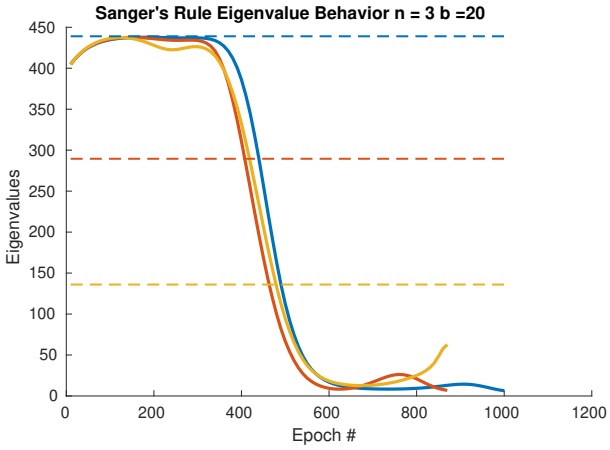
And also for the first three components, in fig. 9.



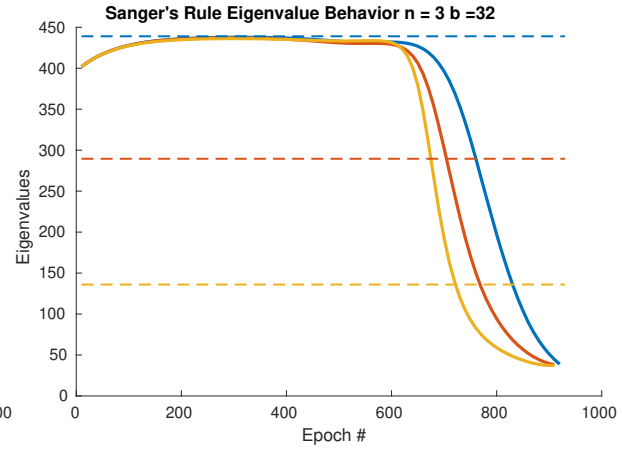
(a) Batch size 10



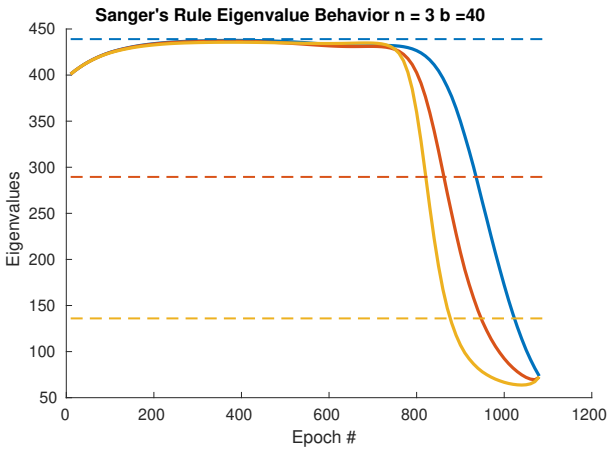
(b) Batch size 16



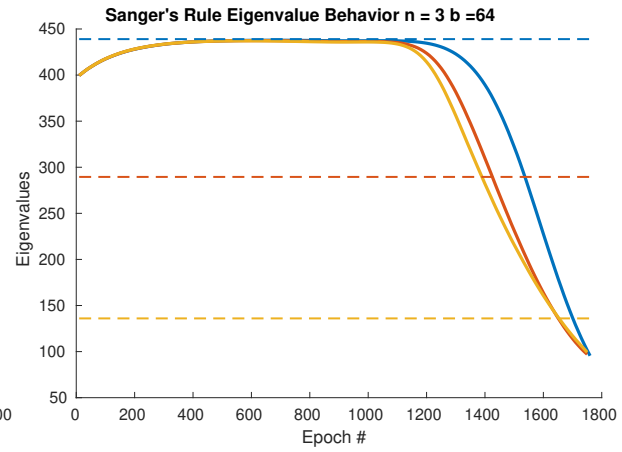
(c) Batch size 20



(d) Batch size 32



(e) Batch size 40



(f) Batch size 64

Figure 9: Sanger's rule convergence behavior for 3 components

After carefully investigating these graphs we can see, that unlike the behavior for the 2D

case given in fig. 6, the eigenvalues do not remain stable. In particular, the higher eigenvalues do not converge, before the first eigenvalue starts to diverge. This is crucial, given that in the implementation in listing 2, we introduce a dependence on the previous results, i.e. previous weights. If they do not manage to remain stable for a time long enough, no convergence for the next ones is possible.

One of the closer compromises is obtained with the parameters:

batch size 40 **iterations** 865 **learning rate** 1e−6

Results are shown in fig. 10.

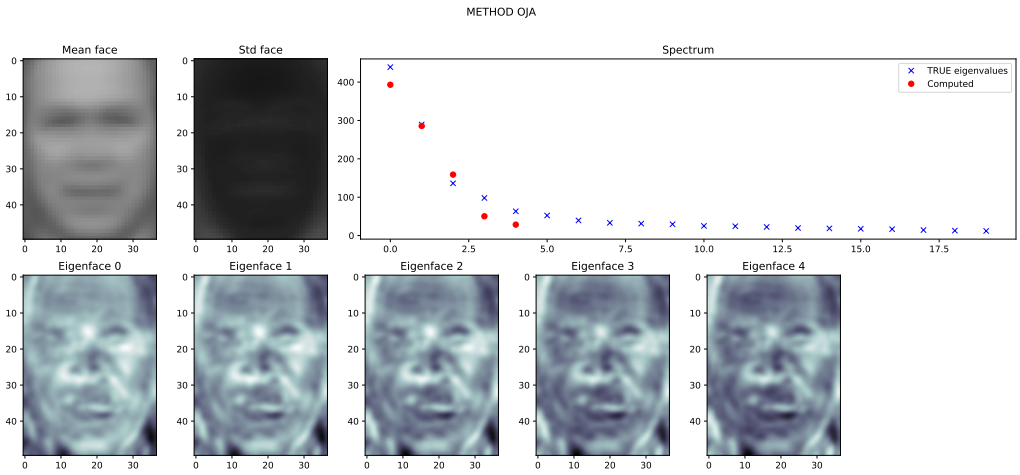


Figure 10: Sanger's faces components results

We can see that the eigenfaces look all quite similar, and the first eigenface already started to diverge from its stable values.

Oja's rule instead converges to the same eigenface, given in fig. 11 (just like Sanger's stopping at few iterations).

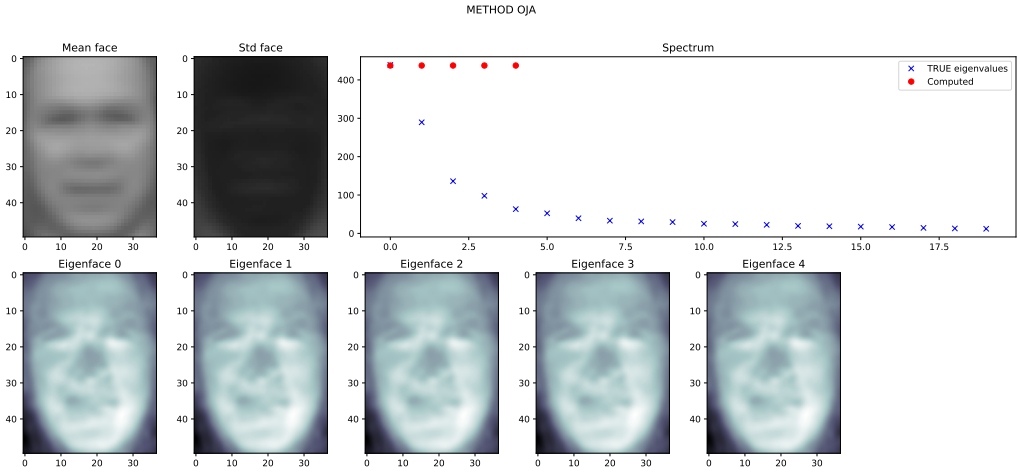


Figure 11: Oja's faces components results

Florian Mahlknecht
fmahlknecht@student.ethz.ch
19-945-351

Change in learning rates did not change the results expect from the divergence speed (slower or faster). The main issue is the missing prevention of divergence of already converged eigenvalues. Ideally found eigenvalues would not diverge anymore, but staying fixed at their value, allowing the next one to stably converge. With a fixed learning rate this cannot be guaranteed. With an adaptive learning rate on the other hand, it eventually takes an infinite amount of time to finally converge.

In conclusion, the algorithm is difficult or impossible to tune for the aim of converging to all eigenvalues (and vectors) at the same time.

fmahlknecht@student.ethz.ch
19-945-351