# mbpls Documentation

*Release 1.0.2*

Andreas Baum and Laurent Vermue

Jun 08, 2019

# TABLE OF CONTENTS

An easy to use Python package for (Multiblock) Partial Least Squares prediction modelling of univariate or multivariate outcomes. Four state of the art algorithms have been implemented and optimized for robust performance on large data matrices. The package has been designed to be able to handle missing data, such that application is straight forward using the commonly known Scikit-learn API and its model selection toolbox.

The documentation is available at https://mbpls.readthedocs.io and elaborate (real-world) Jupyter Notebook examples can be found at https://github.com/DTUComputeStatisticsAndDataAnalysis/MBPLS/tree/master/examples

This package can be cited using the following reference.

*Baum et al., (2019). Multiblock PLS: Block dependent prediction modeling for Python. Journal of Open Source Software, 4(34), 1190*

# CHAPTER ONE

# INSTALLATION

- Install the package for Python3 using the following command. Some dependencies might require an upgrade (scikit-learn, numpy and scipy).

  ```
  $ pip install mbpls
  ```

- Now you can import the MBPLS class by typing

  ```
  from mbpls.mbpls import MBPLS
  ```

CHAPTER

TWO

# QUICK START

## 2.1 Use the mbpls package for Partial Least Squares (PLS) prediction modeling

```
import numpy as np
from mbpls.mbpls import MBPLS

num_samples = 40
num_features = 200

# Generate random data matrix X
x = np.random.rand(num_samples, num_features)

# Generate random reference vector y
y = np.random.rand(num_samples,1)

# Establish prediction model using 2 latent variables (components)
pls = MBPLS(n_components=2)
pls.fit(x,y)
y_pred = pls.predict(x)
```

## 2.2 The mbpls package for Multiblock Partial Least Squares (MB-PLS) prediction modeling

```
import numpy as np
from mbpls.mbpls import MBPLS

num_samples = 40
num_features_x1 = 200
num_features_x2 = 250

# Generate two random data matrices X1 and X2 (two blocks)
x1 = np.random.rand(num_samples, num_features_x1)
x2 = np.random.rand(num_samples, num_features_x2)

# Generate random reference vector y
y = np.random.rand(num_samples, 1)

# Establish prediction model using 3 latent variables (components)
```

(continues on next page)

(continued from previous page)

```
mbpls = MBPLS(n_components=3)
mbpls.fit([x1, x2],y)
y_pred = mbpls.predict([x1, x2])

# Use built-in plot method for exploratory analysis of multiblock pls models
mbpls.plot(num_components=3)
```

### 2.2.1 MB-PLS introduction example

*This example refers to the* **mbpls** *package for Python (https://pypi.org/project/mbpls/ ).*

The notebook intends to illustrate how to use Multiblock Partial Least Squares (MB-PLS) regression. To make things easier we are going to use a very simple simulated dataset with two **X** blocks.

MB-PLS aims at establishing predictive models using latent variable spaces. In addition to PLS, it provides a measure on how much each **X** block contributes to the actual prediction of the response **Y**.

#### First, we initialize our data simulation by defining the parameters below.

Let's start without noise. Once you have run through this notebook just go ahead and increase the noise. It will give you a feeling for the influence of noise on the MBPLS estimation.

[1]:

```
rand_seed = 25
num_samples = 20
num_vars_x1 = 25
num_vars_x2 = 45
noise = 0      # add noise between 0..10
```

[2]:

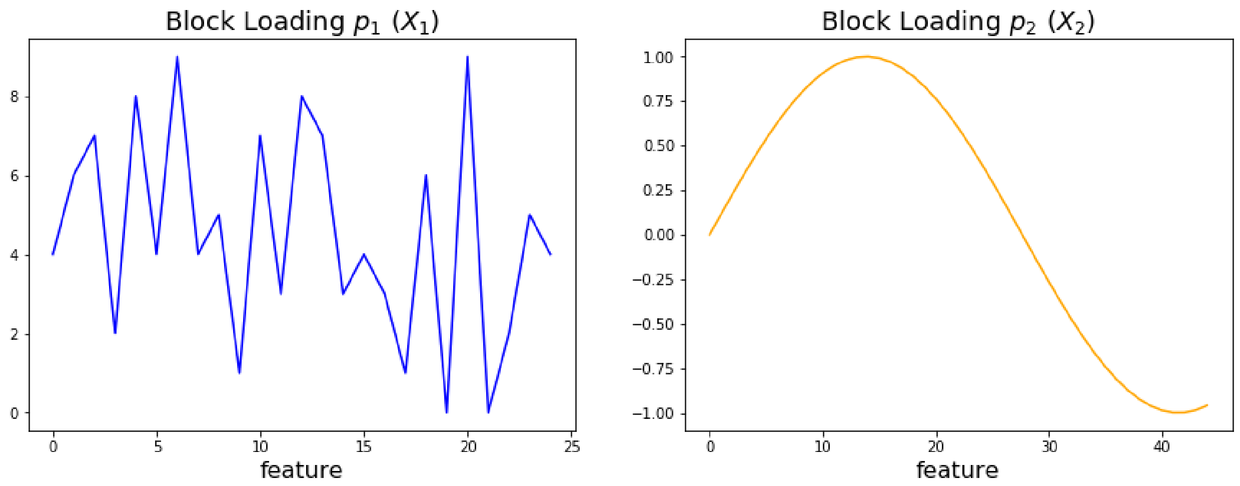
```
from matplotlib import pyplot as plt
import numpy as np
from scipy.stats import ortho_group
```

#### Generate loadings

To generate some data we define two loading vectors, which we will utilize to generate data for $\mathbf{X}_1$ and $\mathbf{X}_2$, respectively. As you can see below these loading vectors have different characterstic shapes. Further down, we will refer to these lodings as the "ground truth".

[3]:

```
np.random.seed(rand_seed)
p1 = np.expand_dims(np.random.randint(0, 10, num_vars_x1), 1)
p2 = np.expand_dims(np.sin(np.linspace(0, 5, num_vars_x2)), 1)

fig, ax = plt.subplots(ncols=2, figsize=(15,5))
ax[0].plot(p1, color='blue')
ax[0].set_title('Block Loading $p_1$ ($X_1$)', fontsize=18)
ax[0].set_xlabel('feature', fontsize=16)
ax[1].plot(p2, color='orange')
ax[1].set_title('Block Loading $p_2$ ($X_2$)', fontsize=18)
ax[1].set_xlabel('feature', fontsize=16);
```

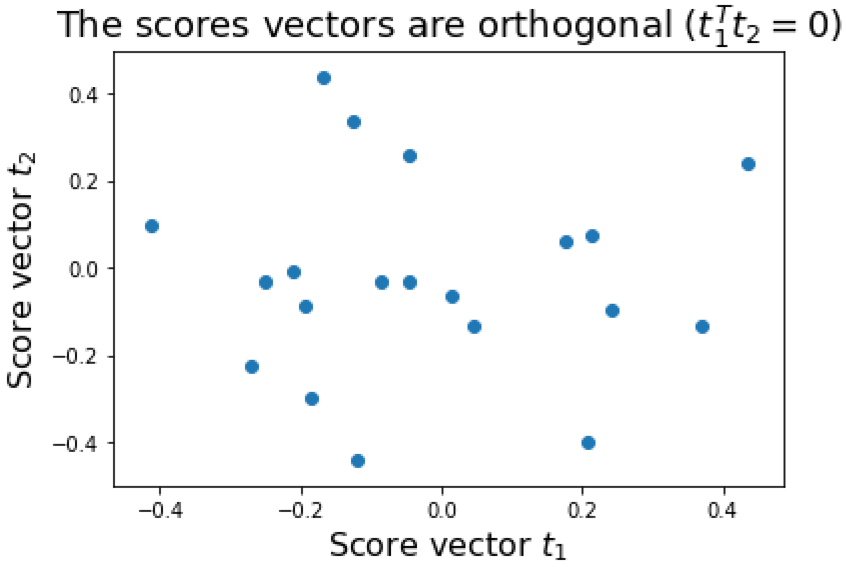


### Generate orthogonal scores

Here we generate some orthogonal scores. We impose orthogonality to make this example clear and simple (no mising of the latent variables). Of course, real-world data would typically deviate from this condition. Further down, we will refer to these scores as the "ground truth" (= $\mathbf{Y}$).

[4]:

```
t = ortho_group.rvs(num_samples, random_state=rand_seed)[:, 0:2]
t1 = t[:,0:1]
t2 = t[:,1:2]

plt.figure()
plt.scatter(t1, t2)
plt.xlabel('Score vector $t_1$', size=16)
plt.ylabel('Score vector $t_2$', size=16)
plt.title('The scores vectors are orthogonal ($t_1^Tt_2 = 0$)', fontsize=18);
```



### Generate data using loadings and scores

Two data blocks are generated. Block $\mathbf{X}_1$ is formed as the outer vector product of loading vector $\mathbf{p}_1$ and score vector $\mathbf{t}_1$. Similarily, the data block $\mathbf{X}_2$ is calculated as the outer vector product of loading vector $\mathbf{p}_2$ and score vector $\mathbf{t}_2$.

$\mathbf{X}_1 = \mathbf{t}_1 \cdot \mathbf{p}_1^T$

$\mathbf{X}_2 = \mathbf{t}_2 \cdot \mathbf{p}_2^T$

[5]:

```
x1 = np.dot(t1, p1.T)
x2 = np.dot(t2, p2.T)
```

### Add noise to the data (according to noise parameter)

Here we add some gaussian noise to show how this impacts the analysis.

[6]:

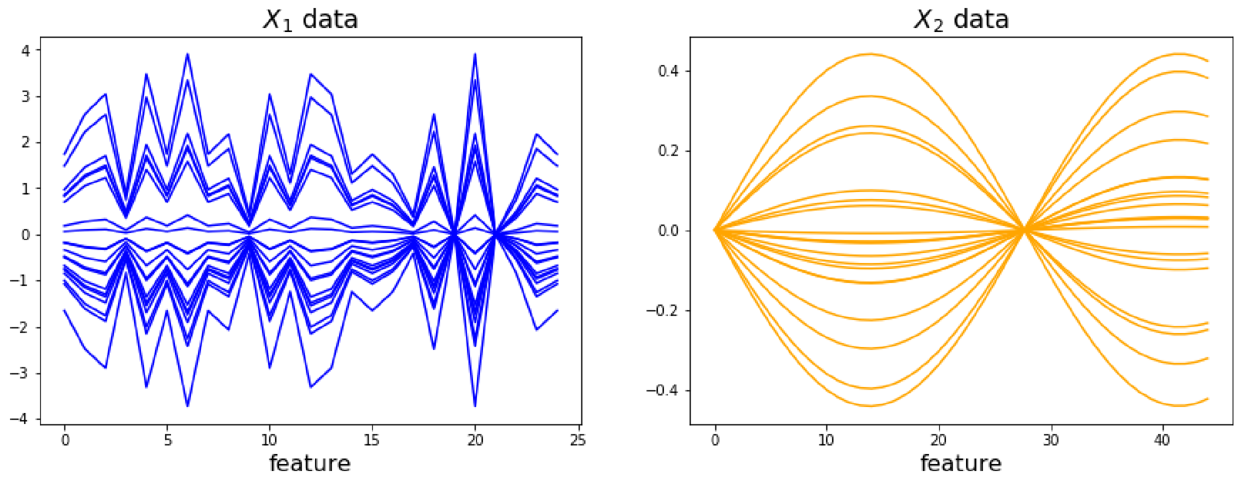
```
x1 = np.random.normal(x1, 0.05*noise)
x2 = np.random.normal(x2, 0.05*noise)
```

### Plot data blocks $\mathrm{X}_1$ and $\mathrm{X}_2$

Let's look at the data blocks $\mathbf{X}_1$ and $\mathbf{X}_2$. The variance in $\mathbf{X}_1$ is related to the first score vector $\mathbf{t}_1$ while the variance in $\mathbf{X}_2$ is related to the second score vector $\mathbf{t}_2$

[7]:

```
fig, ax = plt.subplots(ncols=2, figsize=(15,5))
ax[0].plot(x1.T, color='blue')
ax[0].set_title('$X_1$ data', fontsize=18)
ax[0].set_xlabel('feature', size=16)
ax[1].plot(x2.T, color='orange')
ax[1].set_title('$X_2$ data', fontsize=18)
ax[1].set_xlabel('feature', size=16);
```



### Perform MB-PLS

We perform MB-PLS to fit the data. As a result we will obtain super scores, block loadings and scores and block importances. For further information on the algorithms please check out reference [1].

With this simple example we aim to find out about the following:

1. How much does each block contribute to the prediction of the score vectors $\mathbf{t}_1$ and $\mathbf{t}_2$ ($= \mathbf{Y}$)? This measure we call block importance $a$.
2. What are the feature contributions in each block (block loadings $\hat{\mathbf{p}}_1$ and $\hat{\mathbf{p}}_2$)?
3. Are the fitted block scores $\hat{\mathbf{t}}_1$ and $\hat{\mathbf{t}}_2$ describing the ground truth of our chosen block scores $\mathbf{t}_1$ and $\mathbf{t}_2$?

[8]:

```
from mbpls.mbpls import MBPLS
mbpls_model = MBPLS(n_components=2, standardize=False)
mbpls_model.fit(X=[x1, x2], Y=t);
```

You can find further information on how the model is fitted to the data here:

https://mbpls.readthedocs.io/en/latest/mbpls.html

In this example we are fitting a MB-PLS model to predict $\mathbf{Y}$ from the two data blocks $\mathbf{X}_1$ and $\mathbf{X}_2$. As $\mathbf{Y}$ contains our two known orthogonal score vectors we expect to require two latent variables (LV) to fit the data.

Usually, we strictly recommend to standardize the data. This is important to give the blocks similar weight on the model, i.e. to correct for different feature variances across blocks. However, in this example we do not use standardization because it would destroy the imposed orthogonality of the above defined scores.

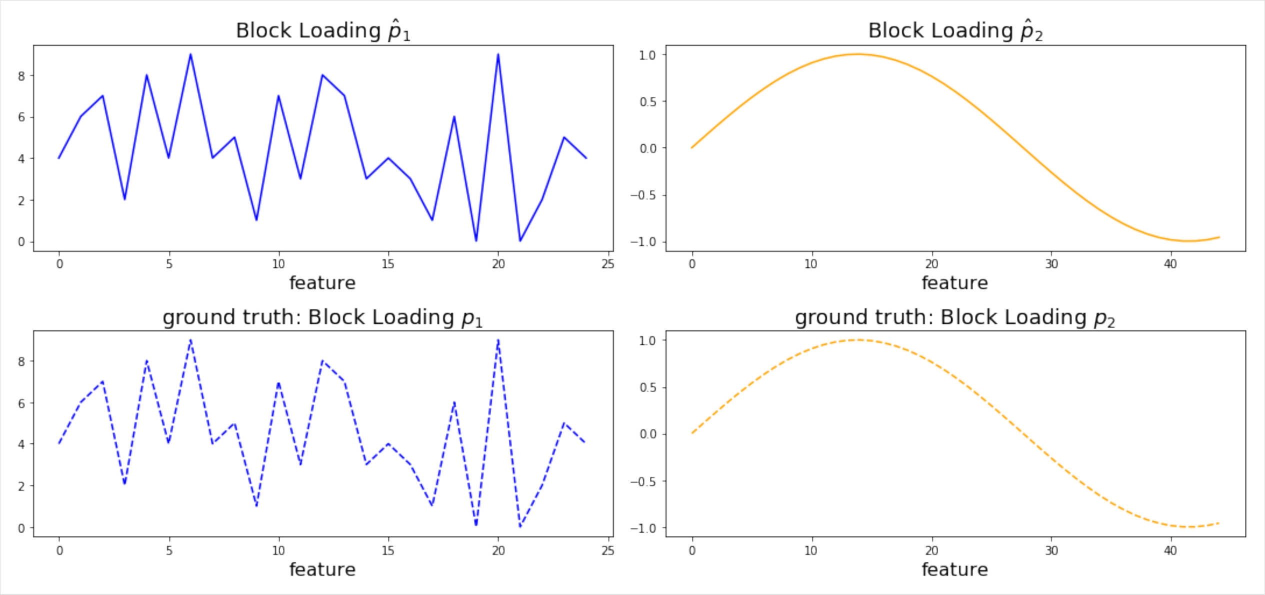
### Plot $\hat{\mathrm{p}}_1$ and $\hat{\mathrm{p}}_2$ and compare to ground truth loadings $\mathrm{p}_1$ and $\mathrm{p}_2$

[9]:

```
p1_hat = mbpls_model.P_[0][:,0]
p2_hat = mbpls_model.P_[1][:,1]
```

[10]:

```
fig, ax = plt.subplots(ncols=2, nrows=2, figsize=(15,7))
ax[0][0].plot(p1_hat, color='blue')
ax[0][0].set_title('Block Loading $\hat{p}_1$', size=18)
ax[0][0].set_xlabel('feature', size=16)
ax[0][1].plot(p2_hat, color='orange')
ax[0][1].set_title('Block Loading $\hat{p}_2$', size=18)
ax[0][1].set_xlabel('feature', size=16)
ax[1][0].plot(p1,color='blue',ls='--')
ax[1][0].set_title('ground truth: Block Loading $p_1$', size=18)
ax[1][0].set_xlabel('feature', size=16)
ax[1][1].plot(p2,color='orange',ls='--')
ax[1][1].set_title('ground truth: Block Loading $p_2$', size=18)
ax[1][1].set_xlabel('feature', size=16)
plt.tight_layout()
```



As you can see above the fitted block loadings $\hat{\mathbf{p}}_1$ and $\hat{\mathbf{p}}_2$ describe our original feature loadings perfectly. Depending on the initialization of the analysis the sign of the loading vectors might switch.

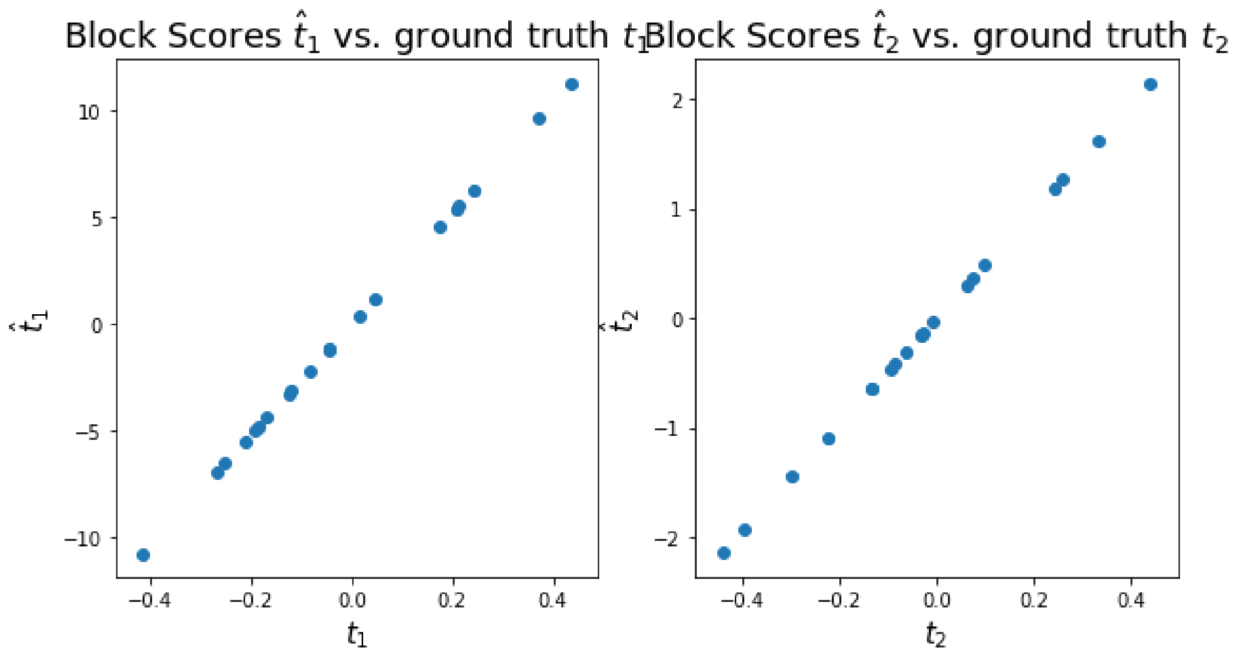
### Block scores $\hat{\mathbf{t}}_1$ in LV1 and $\hat{\mathbf{t}}_2$ in LV2 are highly correlated to ground truth $\mathbf{t}_1$ and $\mathbf{t}_2$

[11]:

```
t1_hat = mbpls_model.T_[0][:,0]
t2_hat = mbpls_model.T_[1][:,1]
```

[12]:

```
fig, ax = plt.subplots(ncols=2, figsize=(10,5))
ax[0].scatter(t1, t1_hat)
ax[0].set_title('Block Scores $\hat{t}_1$ vs. ground truth $t_1$', fontsize=18)
ax[0].set_xlabel('$t_1$', size=15)
ax[0].set_ylabel('$\hat{t}_1$', size=15)
ax[1].scatter(t2, t2_hat)
ax[1].set_title('Block Scores $\hat{t}_2$ vs. ground truth $t_2$', fontsize=18)
ax[1].set_xlabel('$t_2$', size=15)
ax[1].set_ylabel('$\hat{t}_2$', size=15);
```



### Explained variance and block importance

To show the importances and explained variance the MB-PLS model has detected, we extract these parameters from the fitted model.

```
[13]: variances_x = mbpls_model.explained_var_xblocks_
      blockimportances = mbpls_model.A_
      variance_y = mbpls_model.explained_var_y_
```

```
[14]: import pandas as pd
      variances_x = pd.DataFrame(data=variances_x.T, columns=['expl. var. X1',
                                  'expl. var. X2'], index=['LV1', 'LV2'])
      variance_y = pd.DataFrame(data=variance_y, columns=['expl. var. Y'],
                            index=['LV1', 'LV2'])
      blockimportances = pd.DataFrame(data=blockimportances.T, columns=[
          'block importance X1', 'block importance X2'], index=['LV1', 'LV2'])
      pd.concat((variances_x, blockimportances, variance_y), axis=1).round(3)
```

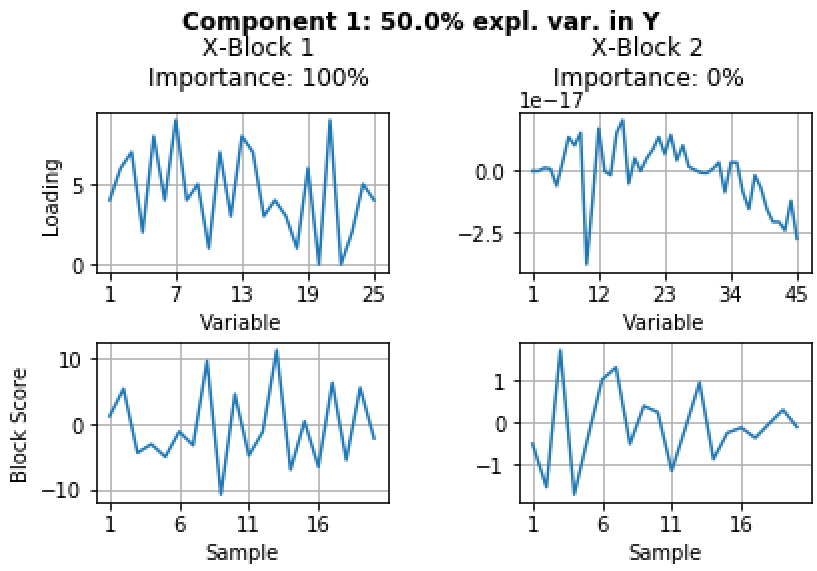
```
[14]:      expl. var. X1  expl. var. X2  block importance X1  block importance X2  \
      LV1            1.0            0.0                  1.0                  0.0
      LV2            0.0            1.0                  0.0                  1.0

           expl. var. Y
      LV1           0.5
      LV2           0.5
```

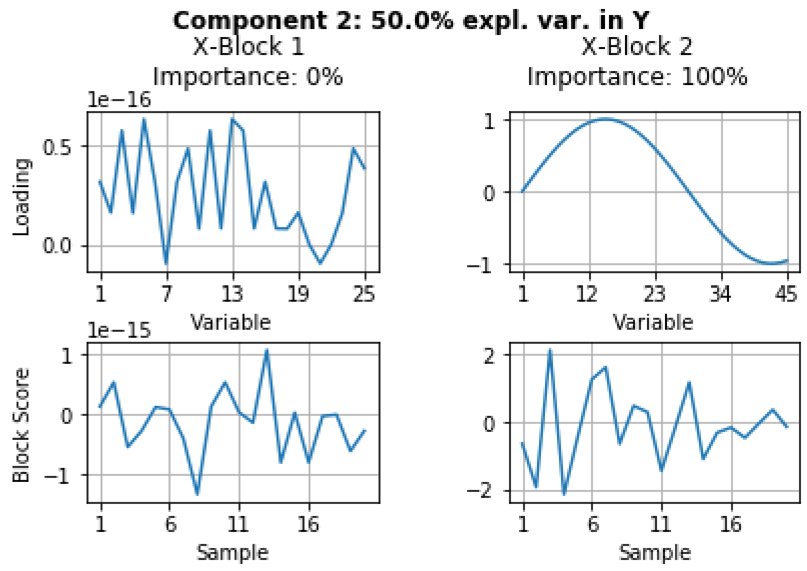
As shown in the table above, the model perfectly fitted LV1 to describe block $\mathbf{X}_1$. Accordingly, it describes correctly that the block importance for LV1 lies 100 % in the corresponding block $\mathbf{X}_1$. Analogue to this, it correctly detected that block $\mathbf{X}_2$ has 100% importance in LV2.

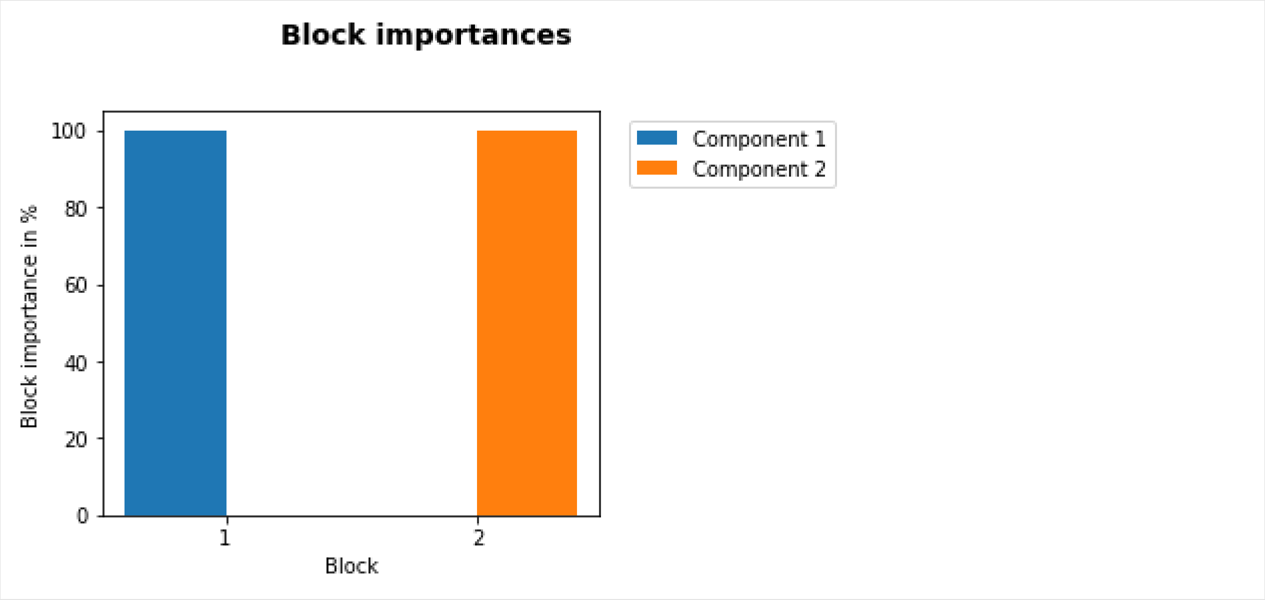
### Using the custom plot function

However, this can be done much easier with the automatic custom visualization through the build in **plot** method of the **mbpls** package, which shows the main fitted attributes, i.e. scores, loadings, explained variance in **Y** and block importances, of the fitted model by calling:

```
[15]: mbpls_model.plot(num_components=2)
```







*Note: In this artificial case the* **block importance** *of both blocks is 0% for one of the respective components/LVs that they do not contribute to and thus there is only one bar for each block.*

### References

[1] J. A. Westerhuis, T. Kourti, and J. F. MacGregor, "Analysis of multiblock and hierarchical PCA and PLS models," J. Chemom., vol. 12, no. 5, pp. 301–321, Sep. 1998.

## 2.2.2 Real World Examples

### Predicting pectin extraction yield from FTIR ($X_1$) and carbohydrate microarray ($X_2$) data using Multiblock Partial Least Squares regression (MB-PLS)

*The data used in this notebook originates from [1]. Further information, analysis and results can be found in the paper. If you re-use and/or re-distribute the data provided please cite the following paper to acknowledge the authorship [1].*

In this example we use Multiblock Partial Least Squares regression to predict the pectin extraction yield for samples, which were measured on FTIR and Carbohydrate Microarrays during the extraction process. The aim is to establish a prediction model and to investigate how much each measurement block contributes to the prediction of the extraction yield. In addition, MB-PLS gives us the opportunity to interpret the prediction model further, i.e. with respect to how different extraction groups relate to this prediction model.

We recommend that you check out the *FTIR_PLS.pynb* and *Carbohydrate_Microarray_PLS.pynb* prior to working on this notebook. These two notebooks elaborate on modeling pectin yield using both data sets independently (single block PLS).

### Load dependencies and data

If you encounter errors in this section you might need to install some of the python packages stated below (or upgrade).

[1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from mbpls.data.get_data import load_Intro_Data
from scipy.io import loadmat
```

[2]:

```
data = load_Intro_Data()
```

```
Following dataset were loaded as Pandas Dataframes:
 dict_keys([’extraction1’, ’ftir1’, ’ftir3’, ’extraction3’, ’ftir2’, ’extraction2’])
```

[3]:

```
# X1 part
ftir1 = data['ftir1']
ftir2 = data['ftir2']
ftir3 = data['ftir3']
# X2 part
carb1 = data['extraction1']
carb2 = data['extraction2']
carb3 = data['extraction3']
```

### Create $X_1$ and $X_2$

To get started with our MB-PLS modeling we merge all dataframes across the sample dimension, meaning that extractions 1, 2 and 3 are merged for FTIR ($\mathbf{X}_1$), carbohydrate microarray ($\mathbf{X}_2$) and pectin yield ($\mathbf{Y}$) data.

[4]:

```
x1 = pd.concat((ftir1, ftir2, ftir3))
x2 = pd.concat((carb1, carb2, carb3))
x2.index = x1.index
y = np.array(x2.index)
wavenumbers = x1.columns
x1x2 = pd.concat((x1, x2), axis=1)
```

### Find number of latent variables using Leave-One-Out Cross Validation

To find the right number of latent variables (LV) we perform cross validation. Scikit-learn makes it easy for us using the cross_val_predict function. We simply re-model the data using different numbers of latent variables and plot the resulting Mean Square Errors (MSE). We pick the number of latent variables where the MSE is minimal to avoid overfitting and to induce as much complexity for our model to obtain most accurate results. Please be aware that the data size is very limited. Therefore, we are risking to overfit. In general, it is recommended to perform 5-fold Cross Validation (CV). However, due to the small sample size we use Leave-One-Out CV.

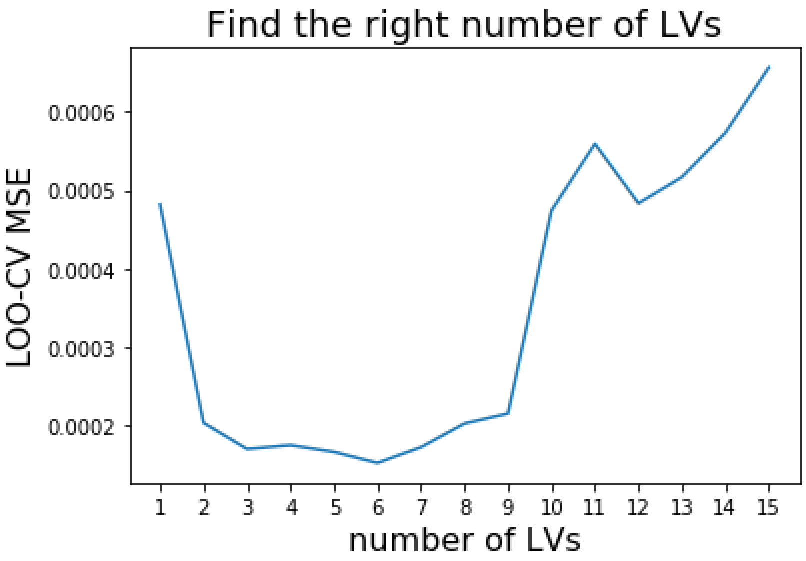
[5]:

```
from mbpls.mbpls import MBPLS
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import mean_squared_error

MSEs = []
for lv in range(15):
    mbpls = MBPLS(n_components=lv+1)
    prediction = cross_val_predict(mbpls, x1x2, y, cv=len(x1x2))
    prediction = pd.DataFrame(prediction)
    MSEs.append(mean_squared_error(prediction, y))
```

(continues on next page)

(continued from previous page)

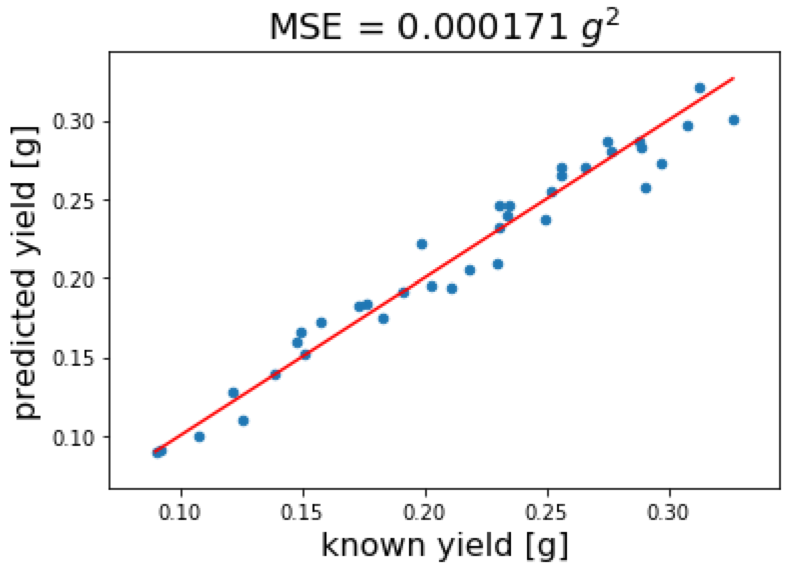
```
plt.plot(np.arange(1,16), MSEs)
plt.xlabel('number of LVs', fontsize=16)
plt.xticks(np.arange(1,16), np.arange(1,16))
plt.ylabel('LOO-CV MSE', fontsize=16)
plt.title('Find the right number of LVs', fontsize=18);
```



### Plot resulting calibration curve and evaluate model

Our plot above indicates that 3 LVs are necessary to establish our MBPLS model. We choose 3 instead of 6 LVs to avoid overfitting due to the small sample size. We re-model using 3 LVs and plot the resulting calibration to evaluate the Mean Square Error (MSE) of cross validation.

[7]:
```
from mbpls.mbpls import MBPLS
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import mean_squared_error

mbpls = MBPLS(n_components=3)
prediction = cross_val_predict(mbpls, x1x2, y, cv=len(x1x2))
prediction = pd.DataFrame(prediction)
y = pd.DataFrame(y)
prediction = pd.concat((prediction, y), axis=1)
prediction.columns=['predicted yield [g]', 'known yield [g]']
prediction.plot.scatter(x='known yield [g]', y='predicted yield [g]')
plt.plot([prediction.min().min(), prediction.max().max()],
         [prediction.min().min(), prediction.max().max()], color='red')
plt.ylabel(prediction.columns[0], fontsize=16)
plt.xlabel(prediction.columns[1], fontsize=16)
plt.title('MSE = {:.6f} $g^2$'.format(mean_squared_error(prediction['known yield [g]
 ↪'],
                                                   prediction['predicted yield [g]'])), fontsize=18);
```



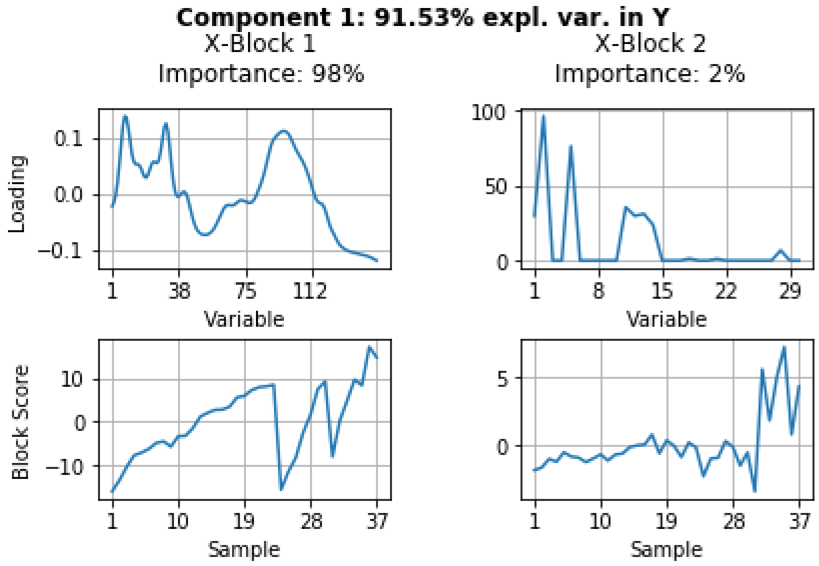
### Interpret Multiblock PLS model

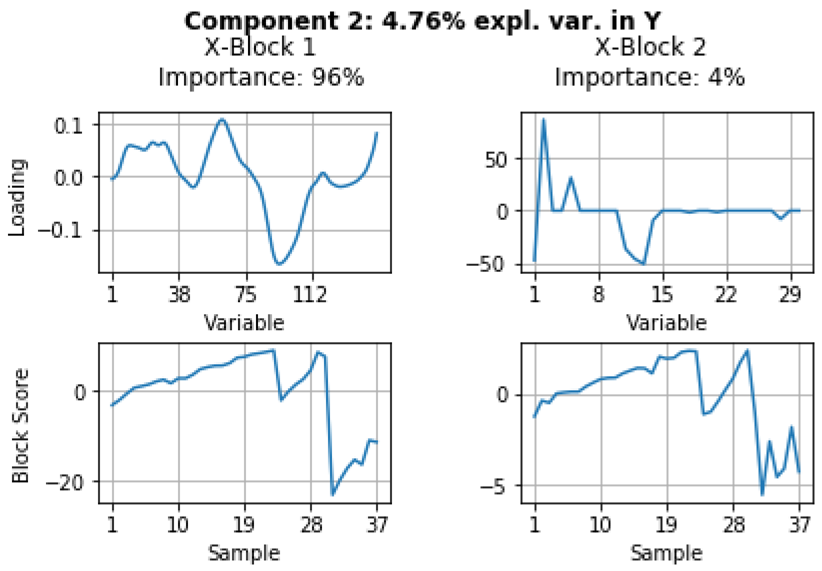
Up to now we have done nothing different than prediction modeling. Although we have used MB-PLS the solution is equivalent to PLS. Further information on how the model is fitted can be found here:

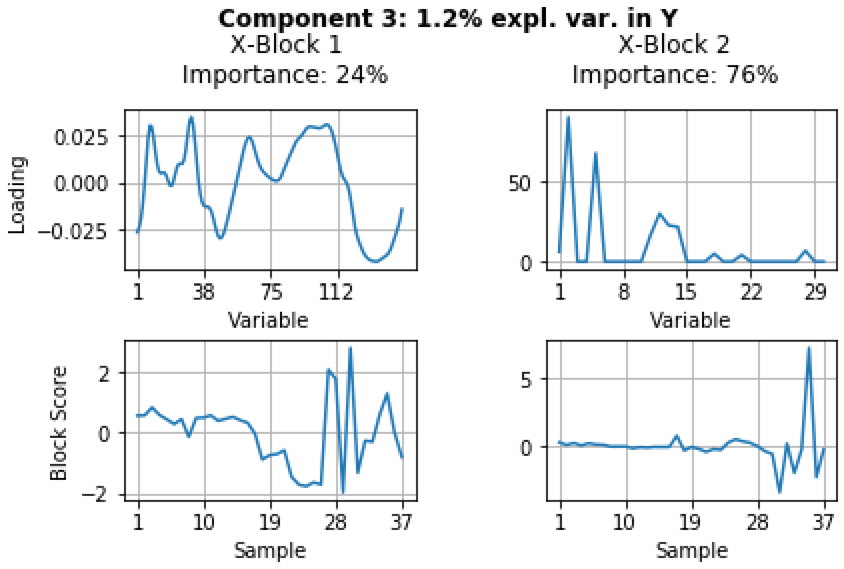
https://mbpls.readthedocs.io/en/latest/mbpls.html

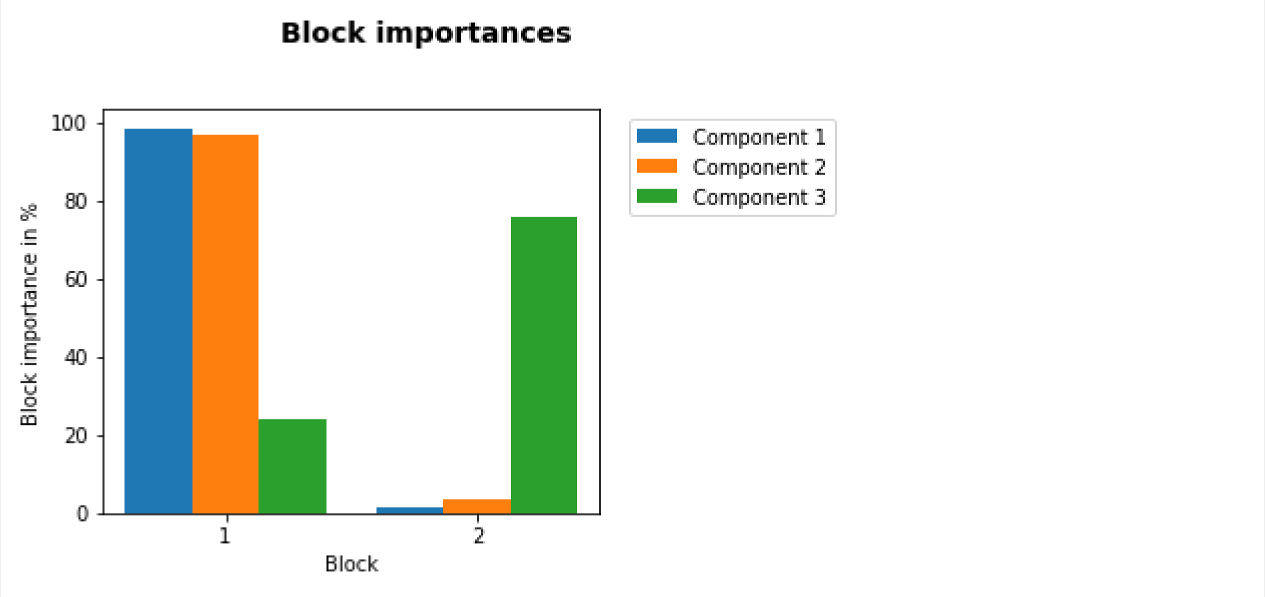
Now, we want to find out more about how each individual block contributes to the prediction of the final pectin yield. To do so we fit an `MBPLS` estimator and look at the results using the built-in `plot` method

```
[8]: mbpls = MBPLS(n_components=3)
     mbpls.fit(X=[x1, x2], Y=y)
     # Plot results for the first 3 components (=LVs)
     mbpls.plot(num_components=3)
```




Component 2: 4.76% expl. var. in Y
X-Block 1
Importance: 96%
X-Block 2
Importance: 4%
Loading
0.1
0.0
−0.1
1
38
75
112
Variable
50
0
−50
1
8
15
22
29
Variable
Block Score
0
−20
1
10
19
28
37
Sample
0
−5
1
10
19
28
37
Sample


Component 3: 1.2% expl. var. in Y
X-Block 1
Importance: 24%
X-Block 2
Importance: 76%
Loading
0.025
0.000
−0.025
1
38
75
112
Variable
50
0
1
8
15
22
29
Variable
Block Score
2
0
−2
1
10
19
28
37
Sample
5
0
1
10
19
28
37
Sample



**The results above show a lot of information. Let's summarize what we can see from these results.**

1. Component 1 (= LV1) explains 92.3% of variance in pectin yield ($\mathbf{Y}$).
2. For the explanation of these 92.3% block $\mathbf{X}_1$ is of 98% and block $\mathbf{X}_2$ of 2% importance.
3. Because block $\mathbf{X}_1$ is of major importance to explain most of the variance in pectin yield we can go ahead and look further at the block Loadings and scores for this component. We merged our data in the sequence extraction 1 (samples 1-23), extraction 2 (samples 24-30) and extraction 3 (samples 31-37). It turns out that the latent variable loading of component one indicates the FTIR spectral regions which explain differences in pectin yield at different time points during extraction.
4. Component 2 explains 3.9% variance in pectin yield. Once again the FTIR block $\mathbf{X}_1$ is of most importance. Looking at the block scores we can see that it differentiates between enzymatic extractions 1 and 2 and acid extraction 3.
5. Component 3 explains only 1.1% of the variance in pectin yield. In this component the carbohydrate microarray data block $\mathbf{X}_2$ contained more important information (78% importance). Looking at the block scores one can find out which samples are highly affected by this component - mainly sample 35 and other samples of the acidic extraction 3. The block loading helps to indentify which of the 30 monoclonal antibodies and/or carbohydrate binding modules were responsible for these samples to vary from the rest.
6. We can conclude that FTIR data is of most importance for prediction of pectin yield. However, the data from the carbohydrate microarrays contribute to the prediction model slightly and can explain minor differences between the different samples. However, these results should be handled carfeully because of the small data size and, hence, insufficient validation of the underlying prediction models.

### Correct block importance for different number of features (variables) in the blocks $\mathrm{X}_1$ and $\mathrm{X}_2$

The block importance measure depends on the number of features in each block. The FTIR data ($\mathbf{X}_1$) contained 143 features, while the carbohydrate microarray data ($\mathbf{X}_2$) contained only 30 features. In such cases it might be desriable to correct the block importance for such differences to represent a more realistic result. With our MBPLS package we have built in a corrected measure which yields the corrected block importance for component $k$ and block $i$ such that:

$BIP_{corr,ik} = BIP_{ik} \cdot (1 - \frac{p_i}{p_{total}})$

In the table below BIPs and block size corrected BIPs are presented for our example.

```
[9]: print('The Block Importances [%] in each latent variable are:')
     BIP = pd.DataFrame(mbpls.A_.round(2), columns=['LV1', 'LV2', 'LV3'],
                        index=['FTIR', 'Carb Micr Array'])
     BIP
```

```
The Block Importances [%] in each latent variable are:
```

```
[9]:                  LV1   LV2   LV3
     FTIR             0.98  0.96  0.24
     Carb Micr Array  0.02  0.04  0.76
```

```
[10]: print('The corrected Block Importances [%] in each latent variable are:')
      BIP_corrected = pd.DataFrame(mbpls.A_corrected_.round(2),
                      columns=['LV1', 'LV2', 'LV3'], index=['FTIR', 'Carb Micr Array'])
      BIP_corrected
```

```
The corrected Block Importances [%] in each latent variable are:
```

```
[10]:                  LV1   LV2   LV3
      FTIR             0.92  0.85  0.06
      Carb Micr Array  0.08  0.15  0.94
```

### Explorative analysis - Let's look a bit closer at the ($X_1$) results

As discussed above we can identify certain patterns in the block scores and loadings. Let's go ahead and look a bit closer at these results. Below you can see how the block scores and loadings are extracted from the mbpls model for further analysis. We scatter the block scores of the three latent variables against each other and color them according to extraction type. Extraction 3 was carried using acid. Again, we can see the difference between acidic and enzymatic pectin extracts. This difference is decribed mostly by latent variable 2. If we look at the loading for LV2 we can identify spectral regions which differentiate between acidic and enzymatic extracted pectin. Further analysis of these results is well in line with the chemical interpretation in [1].

The block scores and loadings shown below are the similar to the results from the *plot* method output. However, we simply aim to exemplify how to extract the data from the model and use it for further ad-hoc analysis. To compare these loadings to the actual FTIR block data we have plotted the spectra for all extractions below as well. The spectral greyscale is resembling the pectin yield (dark - lowest; white - highest)

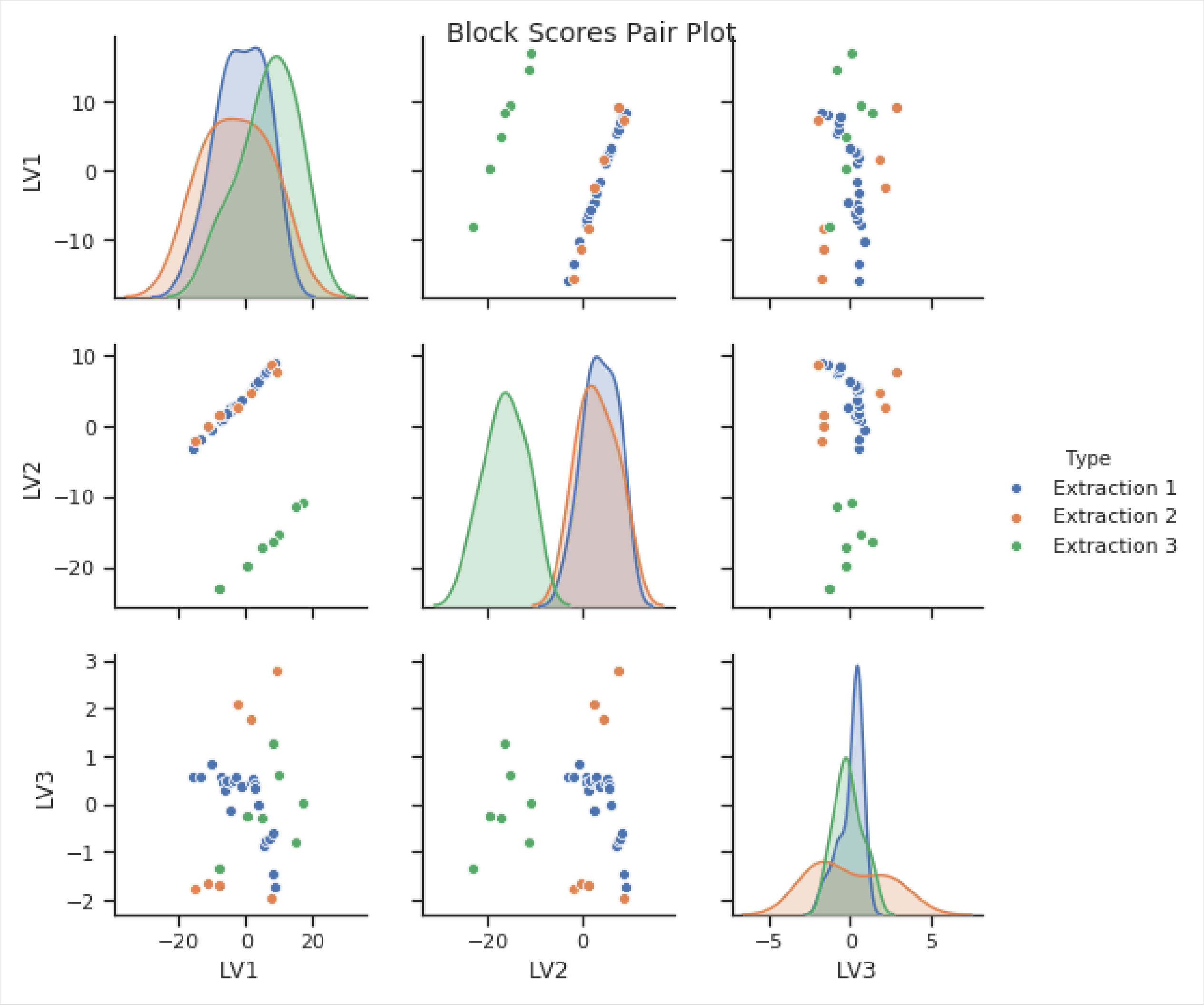
```
[11]: # extract FTIR (X1) block scores from fitted mbpls model
      block_scores_x1 = mbpls.T_[0]

      # extract FTIR (X1) loadings from fitted mbpls model
      block_loadings_x1 = mbpls.P_[0].T

      # inverse transform loadings if standardize option was used during fitting
      # (by default)
      block_loadings_x1 = np.hstack([mbpls.x_scalers_[0].inverse_transform(loading)
                                    .reshape(-1, 1) for loading in block_loadings_x1])

      # Generate a vector which indicates color type for each sample according to
      # extraction 1, 2 and 3
      import itertools
      colors = [['Extraction 1']*23, ['Extraction 2']*7, ['Extraction 3']*7]
      colors = list(itertools.chain(*colors))
      colors = pd.DataFrame(colors, columns=['Type'])

      # Generate pandas dataframes for easy plotting using seaborn
```

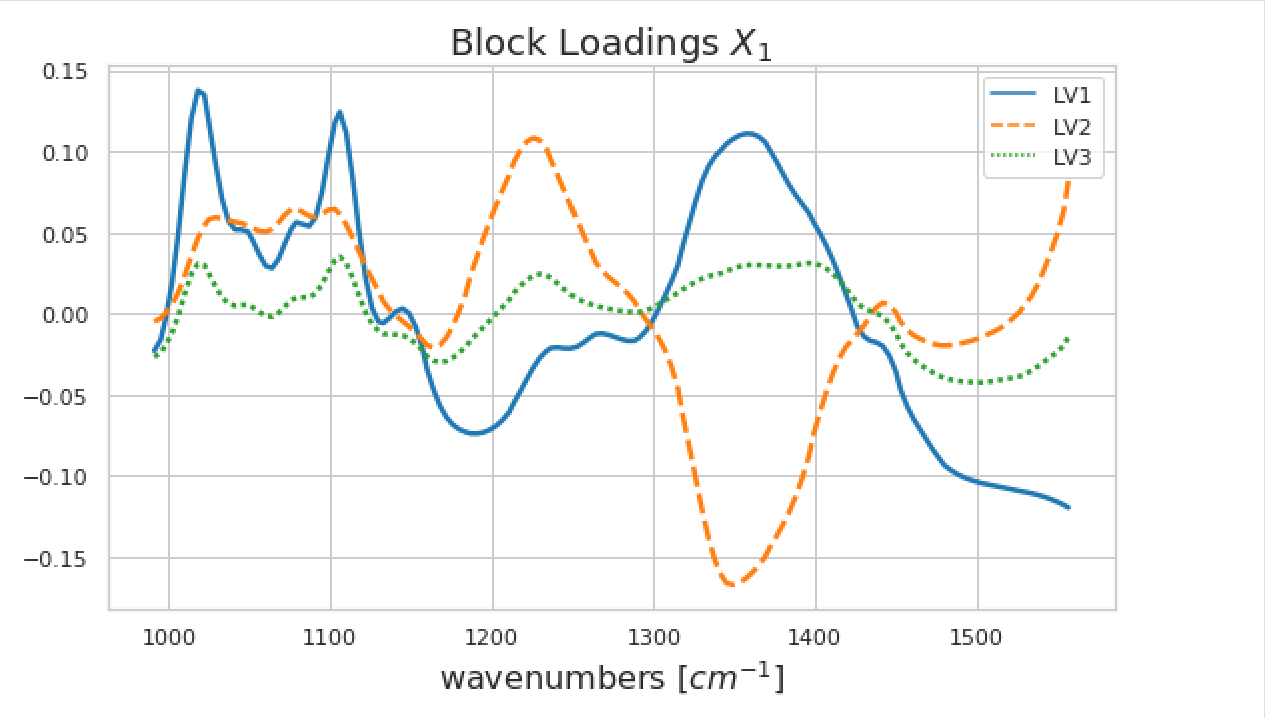
(continues on next page)

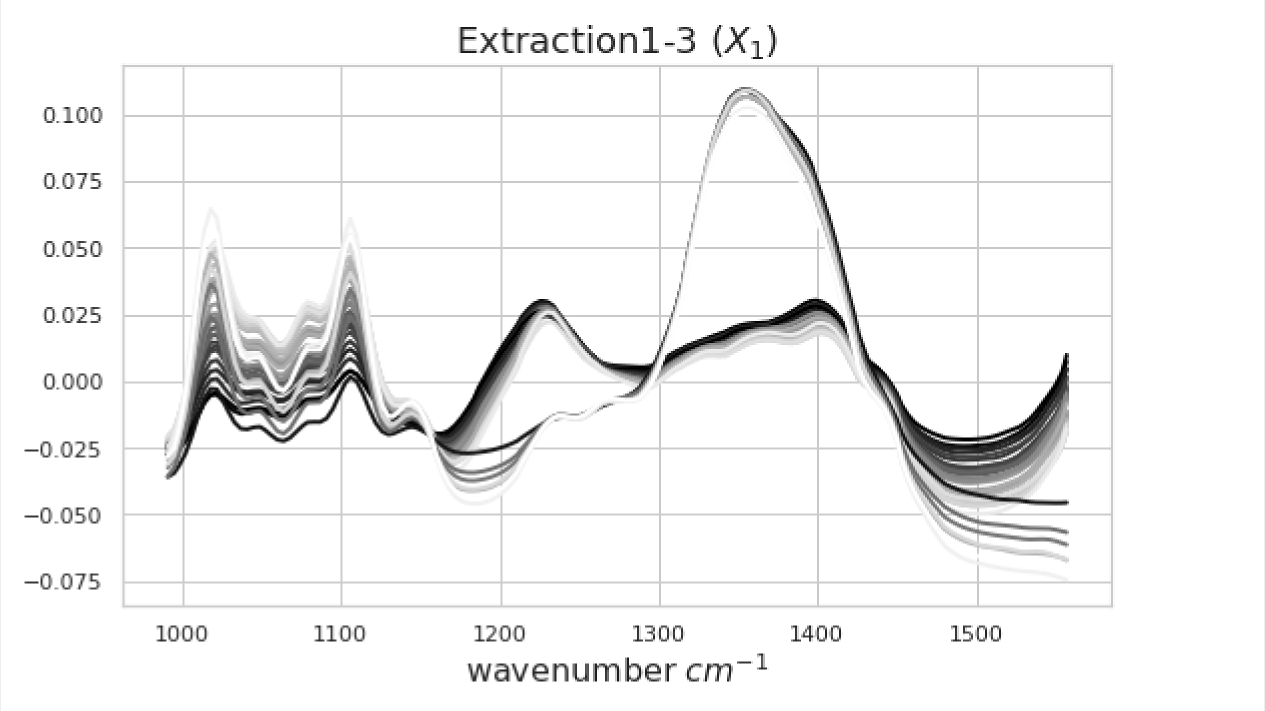
(continued from previous page)

```
block_scores_x1 = pd.DataFrame(block_scores_x1, columns=['LV1', 'LV2', 'LV3'])
block_scores_x1 = pd.concat((block_scores_x1, colors), axis=1)
block_loadings_x1 = pd.DataFrame(block_loadings_x1,
                            columns=['LV1', 'LV2', 'LV3'], index=wavenumbers)

# Plot block scores using pairs plot
import seaborn as sns
sns.set(style="ticks")
sns.pairplot(block_scores_x1, hue="Type")
plt.suptitle('Block Scores Pair Plot')
plt.figure(figsize=(9,5))
sns.set(style="whitegrid")

# Plot FTIR (X1) loadings
sns.lineplot(data=block_loadings_x1, palette="tab10", linewidth=2.5)
plt.title('Block Loadings $X_1$', fontsize=18)
plt.xlabel('wavenumbers [$cm^{-1}$]', fontsize=16);

# Plot function to plot FTIR spectra and color them according extraction
# yield
def plot_spectra(spectra, ax, name):
    pectin_yield = np.array(spectra.index)
    color_code = (pectin_yield - pectin_yield.min())
    color_code = color_code / color_code.max()
    color_code = color_code[:]
    for spectrum, color in zip(np.array(spectra), color_code):
        ax.plot(wavenumbers, spectrum, color=(color, color, color),
                linewidth=2)
        ax.set_title(name, fontsize=18)
        ax.set_xlabel('wavenumber $cm^{-1}$', fontsize=16)
        plt.tight_layout()

# Plot FTIR spectra in groups of extractions for explorative data
# analysis
fig, ax = plt.subplots(figsize=(8,5))
plot_spectra(x1, ax, 'Extraction1-3 ($X_1$)')
```

```
/usr/local/lib/python3.5/dist-packages/scipy/stats/stats.py:1713: FutureWarning:␣
↪Using a non-tuple sequence for multidimensional indexing is deprecated; use␣
↪`arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as␣
↪an array index, `arr[np.array(seq)]`, which will result either in an error or a␣
↪different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```


Block Scores Pair Plot
LV1
LV2
LV3
10
0
−10
10
0
−10
−20
3
2
1
0
−1
−2
−20
0
20
−20
0
−5
0
5
Type
● Extraction 1
● Extraction 2
● Extraction 3





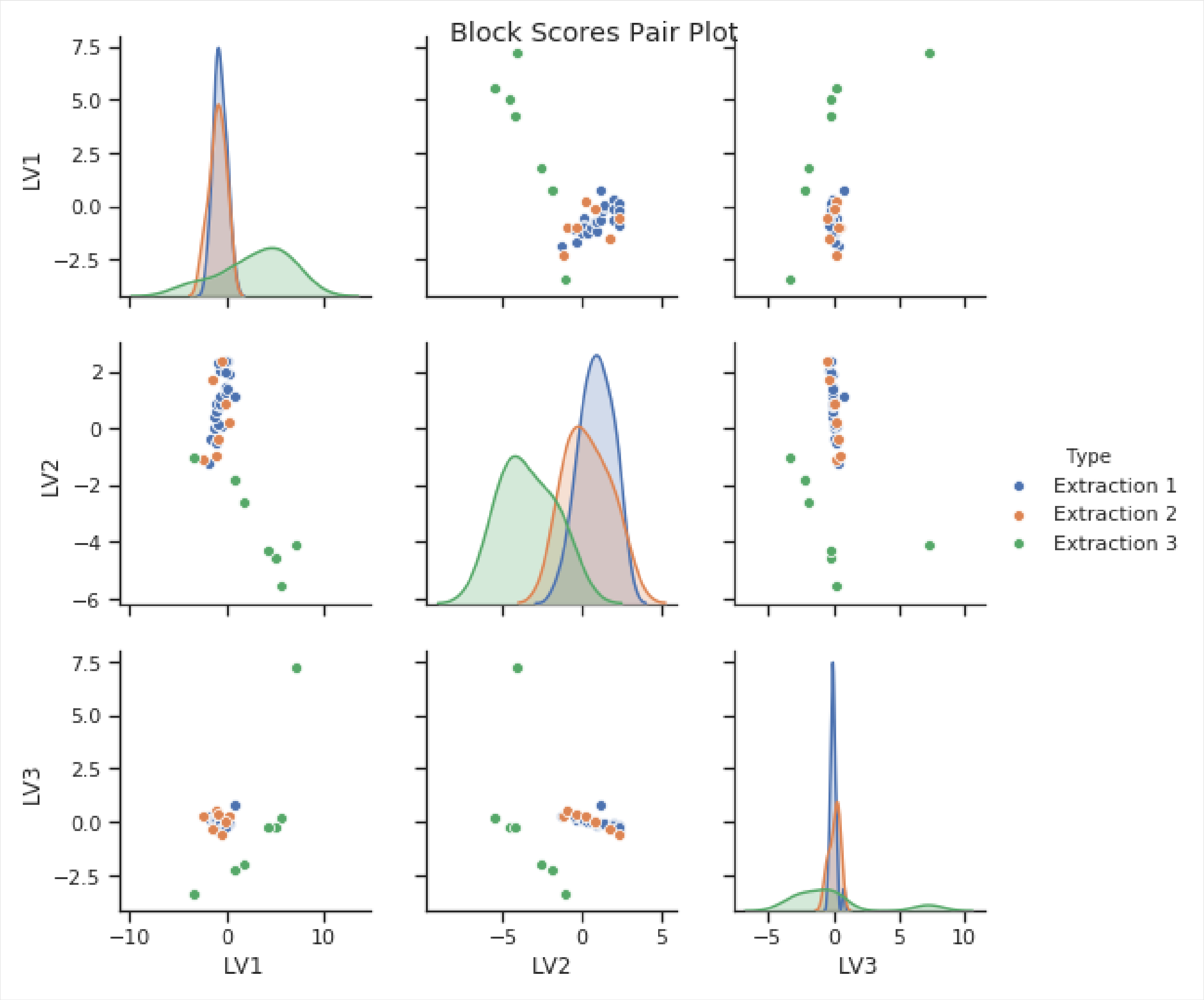
### Explorative analysis - Let's look closer at the $X_2$ results

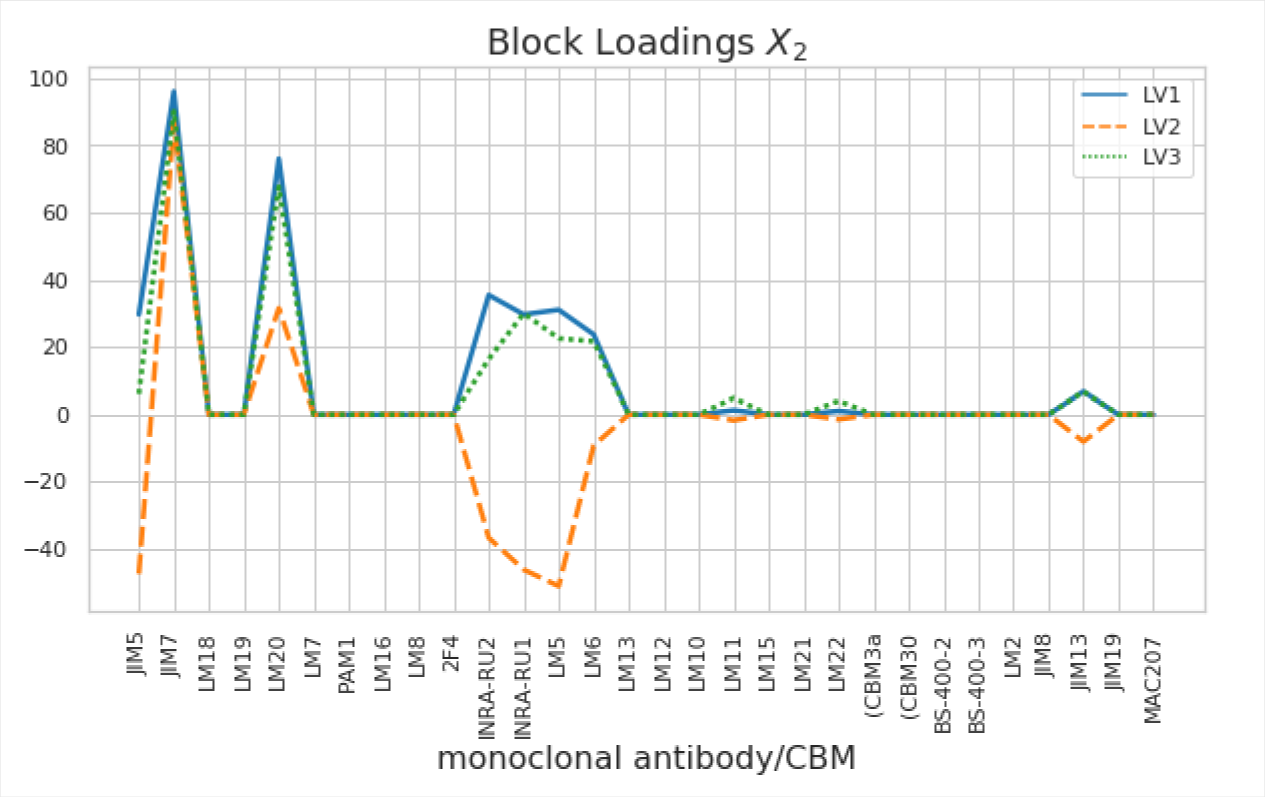
We perform the same analysis as above, but this time for the carbohydrate microarray block ($\mathbf{X}_2$). Above we said that Block $\mathbf{X}_2$ was most important in LV3. When we look at block scores and loadings for LV3 we can see that one sample of extraction 3 performed as an outlier. This sample could have been measured incorrectly. We can also see that all samples from extraction 1 and 2 (enzymatic) were not really affected by LV3 because they lay all close together and indicate scores around zero in LV3. LV3 was therefore most important for the samples from acidic extraction 3. However, we want to keep in mind that LV3 explains only 1.2% variance in pectin yield.

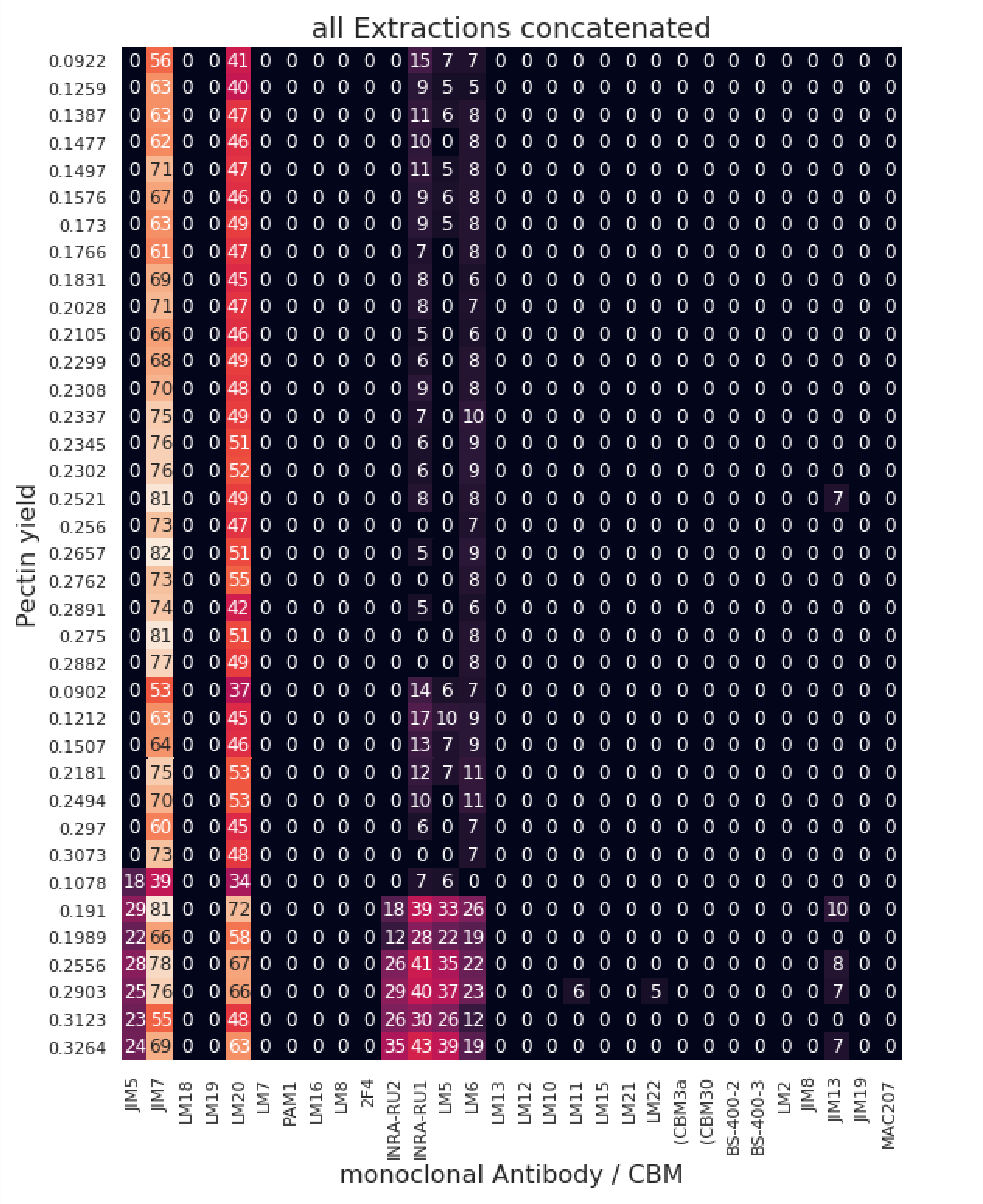
[12]:

```
# extract Carb. Micr. Array (X2) block scores from fitted mbpls model
block_scores_x2 = mbpls.T_[1]

# extract Carb. Micr. (X2) loadings from fitted mbpls model
block_loadings_x2 = mbpls.P_[1].T

# inverse transform loadings if standardize option was used during fitting
# (by default)
block_loadings_x2 = np.hstack([mbpls.x_scalers_[1].inverse_transform(loading)
                            .reshape(-1, 1) for loading in block_loadings_x2])

# Generate pandas dataframes for easy plotting using seaborn
import itertools
colors = [['Extraction 1']*23, ['Extraction 2']*7, ['Extraction 3']*7]
colors = list(itertools.chain(*colors))
colors = pd.DataFrame(colors, columns=['Type'])

block_scores_x2 = pd.DataFrame(block_scores_x2, columns=['LV1', 'LV2', 'LV3'])
block_scores_x2 = pd.concat((block_scores_x2, colors), axis=1)
block_loadings_x2 = pd.DataFrame(block_loadings_x2,
                            columns=['LV1', 'LV2', 'LV3'], index=x2.columns)

# Plot block scores using pairs plot
import seaborn as sns
sns.set(style="ticks")
sns.pairplot(block_scores_x2, hue="Type")
plt.suptitle('Block Scores Pair Plot')

# Plot Carb. Micr. Array (X2) loadings
plt.figure(figsize=(10,5))
sns.set(style="whitegrid")
sns.lineplot(data=block_loadings_x2, palette="tab10", linewidth=2.5,
             sort=False)
plt.xticks(rotation=90)
plt.title('Block Loadings $X_2$', fontsize=18)
plt.xlabel('monoclonal antibody/CBM', fontsize=16);

# Plot Carb. Micr. Array data for comparison and exploratory data analysis
plt.figure(figsize=(9,12))
sns.heatmap(x2, annot=True, cbar=False)
plt.title('all Extractions concatenated', fontsize=18)
plt.xlabel('monoclonal Antibody / CBM', fontsize=16)
plt.ylabel('Pectin yield', fontsize=16);
```

```
/usr/local/lib/python3.5/dist-packages/scipy/stats/stats.py:1713: FutureWarning:␣
↪Using a non-tuple sequence for multidimensional indexing is deprecated; use␣
↪`arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as␣
↪an array index, `arr[np.array(seq)]`, which will result either in an error or a␣
↪different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```


Block Scores Pair Plot
LV1
LV2
LV3
7.5
5.0
2.5
0.0
−2.5
2
0
−2
−4
−6
−10
0
10
−5
5
Type
● Extraction 1
● Extraction 2
● Extraction 3


Block Loadings $X_2$
100
80
60
40
20
0
−20
−40
LV1
LV2
LV3
JIM5
JIM7
LM18
LM19
LM20
LM7
PAM1
LM16
LM8
2F4
INRA-RU2
INRA-RU1
LM5
LM6
LM13
LM12
LM10
LM11
LM15
LM21
LM22
(CBM3a
(CBM30
BS-400-2
BS-400-3
LM2
JIM8
JIM13
JIM19
MAC207
monoclonal antibody/CBM



all Extractions concatenated

| Pectin yield | JIM5 | JIM7 | LM18 | LM19 | LM20 | LM7 | PAM1 | LM16 | LM8 | 2F4 | INRA-RU2 | INRA-RU1 | LM5 | LM6 | LM13 | LM12 | LM10 | LM11 | LM15 | LM21 | LM22 | (CBM3a | (CBM30 | BS-400-2 | BS-400-3 | LM2 | JIM8 | JIM13 | JIM19 | MAC207 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.0922 | 0 | 56 | 0 | 0 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 15 | 7 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.1259 | 0 | 63 | 0 | 0 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 5 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.1387 | 0 | 63 | 0 | 0 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 11 | 6 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.1477 | 0 | 62 | 0 | 0 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.1497 | 0 | 71 | 0 | 0 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 11 | 5 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.1576 | 0 | 67 | 0 | 0 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 6 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.173 | 0 | 63 | 0 | 0 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 5 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.1766 | 0 | 61 | 0 | 0 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.1831 | 0 | 69 | 0 | 0 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.2028 | 0 | 71 | 0 | 0 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.2105 | 0 | 66 | 0 | 0 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.2299 | 0 | 68 | 0 | 0 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.2308 | 0 | 70 | 0 | 0 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.2337 | 0 | 75 | 0 | 0 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.2345 | 0 | 76 | 0 | 0 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.2302 | 0 | 76 | 0 | 0 | 52 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.2521 | 0 | 81 | 0 | 0 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 |
| 0.256 | 0 | 73 | 0 | 0 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.2657 | 0 | 82 | 0 | 0 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.2762 | 0 | 73 | 0 | 0 | 55 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.2891 | 0 | 74 | 0 | 0 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.275 | 0 | 81 | 0 | 0 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.2882 | 0 | 77 | 0 | 0 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.0902 | 0 | 53 | 0 | 0 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 14 | 6 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.1212 | 0 | 63 | 0 | 0 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 17 | 10 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.1507 | 0 | 64 | 0 | 0 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 13 | 7 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.2181 | 0 | 75 | 0 | 0 | 53 | 0 | 0 | 0 | 0 | 0 | 0 | 12 | 7 | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.2494 | 0 | 70 | 0 | 0 | 53 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.297 | 0 | 60 | 0 | 0 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.3073 | 0 | 73 | 0 | 0 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.1078 | 18 | 39 | 0 | 0 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.191 | 29 | 81 | 0 | 0 | 72 | 0 | 0 | 0 | 0 | 0 | 18 | 39 | 33 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 0 |
| 0.1989 | 22 | 66 | 0 | 0 | 58 | 0 | 0 | 0 | 0 | 0 | 12 | 28 | 22 | 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.2556 | 28 | 78 | 0 | 0 | 67 | 0 | 0 | 0 | 0 | 0 | 26 | 41 | 35 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 0 |
| 0.2903 | 25 | 76 | 0 | 0 | 66 | 0 | 0 | 0 | 0 | 0 | 29 | 40 | 37 | 23 | 0 | 0 | 0 | 6 | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 |
| 0.3123 | 23 | 55 | 0 | 0 | 48 | 0 | 0 | 0 | 0 | 0 | 26 | 30 | 26 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.3264 | 24 | 69 | 0 | 0 | 63 | 0 | 0 | 0 | 0 | 0 | 35 | 43 | 39 | 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 |

monoclonal Antibody / CBM

### References

[1] Baum, Andreas, et al. “Prediction of pectin yield and quality by FTIR and carbohydrate microarray analysis.” Food and Bioprocess Technology 10.1 (2017): 143-154.

### PLS for prediction of pectin yield from carbohydrate microarray data

This notebook exemplifies the PLS modeling approach utilizing carbohydrate microarray data. The data used for this notebook was derived from Baum et al. [1]. Carbohydrate microarrays are typically equipped with multiple monoclonal antibodies and carbohydrate binding modules (CBM), which will specifically bind to unique structural polysaccharide domains. Pectin is such a polysaccharide. It is typically extracted from citrus peel and used as a gelling agent in jams etc. In the following we will establish Partial Least Squares (PLS) prediction models for pectin yield from the underlying carbohydrate binding pattern of various extracted samples. Or said in a different way we want to find a PLS model, which we could use on a newly measured carbydrate microarray row (= pectin extract sample) to predict its final pectin yield.

### Import dependencies and load data

If you encounter errors in this section you might need to install/upgrade one or several of the following packages.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from mbpls.data.get_data import load_CarbohydrateMicroarrays_Data
```

[2]:
```
data = load_CarbohydrateMicroarrays_Data()
```

```
Following dataset were loaded as Pandas Dataframes:
 dict_keys([’extraction2’, ’extraction3’, ’extraction1’])
```

[3]:
```
extraction1 = data['extraction1']
extraction2 = data['extraction2']
extraction3 = data['extraction3']
```

### Plot carbohydrate microarray data

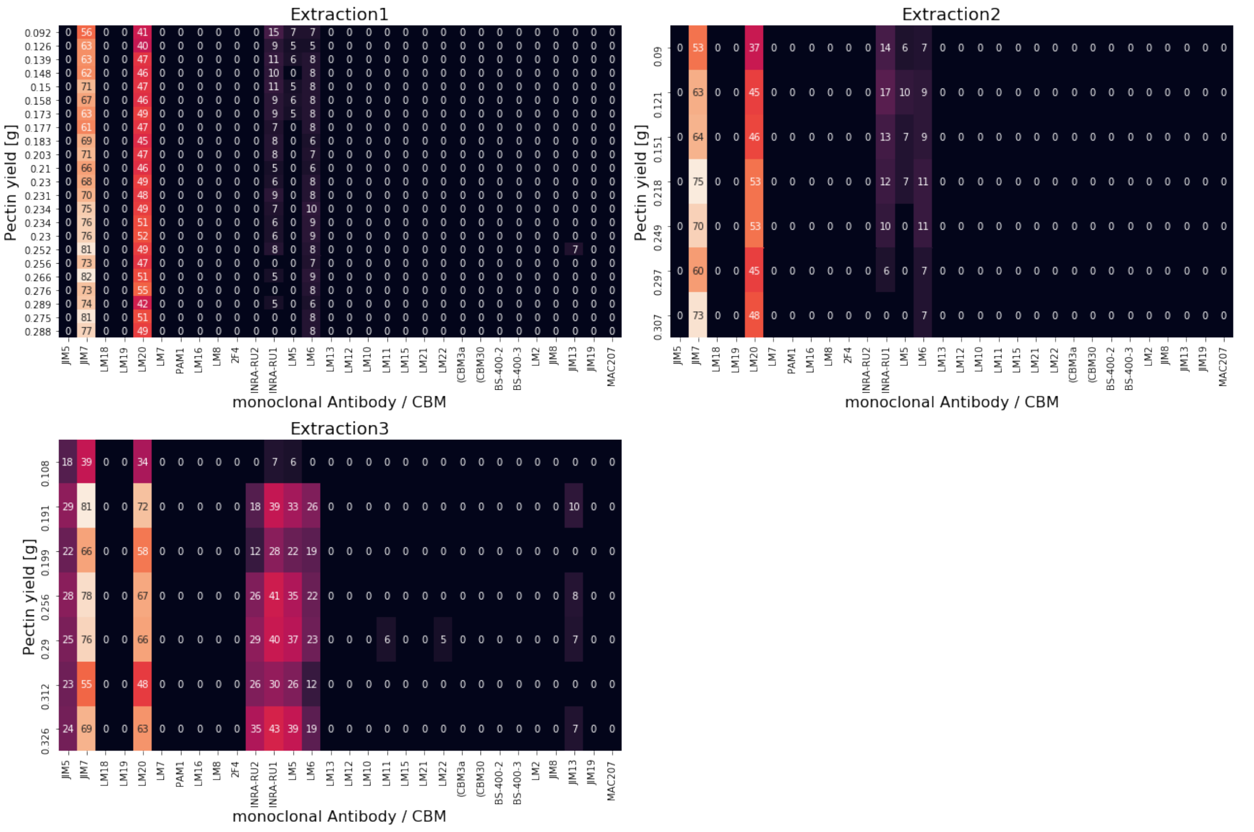
During the study by Baum et al. [1] three extractions of pectin were carried out. These are shown as heatmaps below. The reference values **y** (final pectin yield) are given as row labels. The extracted samples were measured against 30 monoclonal antibodies/CBMs (columns). Lighter color in the individual cells resemble high degree of interaction between antibody/CBM with the extracted pectin.

[4]:
```
fig, ax = plt.subplots(ncols=2, nrows=2, figsize=(18,12))
sns.heatmap(extraction1, ax=ax[0][0], annot=True, cbar=False)
sns.heatmap(extraction2, ax=ax[0][1], annot=True, cbar=False)
sns.heatmap(extraction3, ax=ax[1][0], annot=True, cbar=False)
ax[1][1].axis('off')

ax[0][0].set_title('Extraction1', fontsize=18)
ax[0][1].set_title('Extraction2', fontsize=18)
ax[1][0].set_title('Extraction3', fontsize=18)
ax[0][0].set_xlabel('monoclonal Antibody / CBM', fontsize=16)
```

(continues on next page)

(continued from previous page)

```
ax[0][1].set_xlabel('monoclonal Antibody / CBM', fontsize=16)
ax[1][0].set_xlabel('monoclonal Antibody / CBM', fontsize=16)
ax[0][0].set_ylabel('Pectin yield [g]', fontsize=16)
ax[0][1].set_ylabel('Pectin yield [g]', fontsize=16)
ax[1][0].set_ylabel('Pectin yield [g]', fontsize=16)
plt.tight_layout()
```

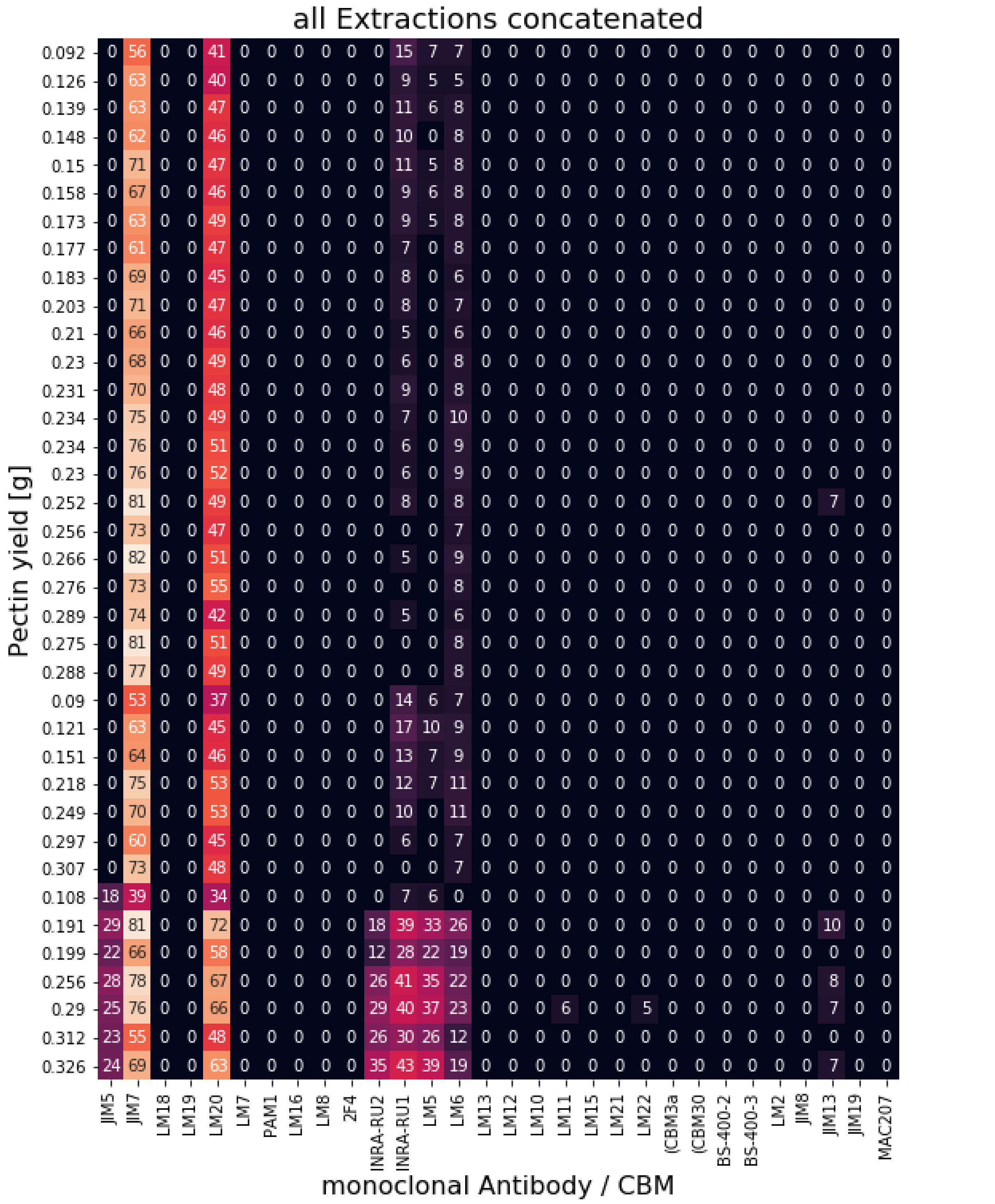


## Concatenate data

To perform PLS modeling all three extractions are concatenated into a single matrix **X** for PLS modeling. Pectin yields resemble reference values **y** for supervised learning.

[5]:
```
extractions_all = pd.concat((extraction1, extraction2, extraction3),axis=0)
yield_all = extractions_all.index
```

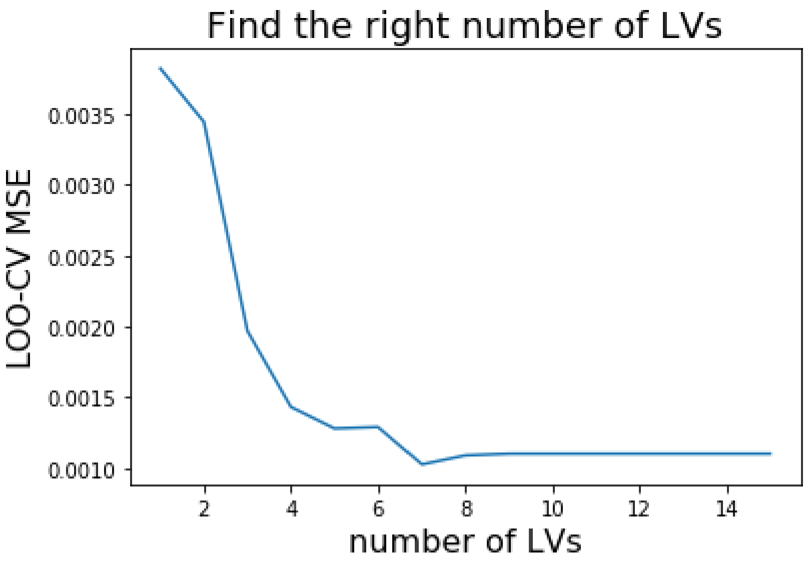
[6]:
```
plt.figure(figsize=(9,12))
sns.heatmap(extractions_all, annot=True, cbar=False)
plt.title('all Extractions concatenated', fontsize=18)
plt.xlabel('monoclonal Antibody / CBM', fontsize=16)
plt.ylabel('Pectin yield [g]', fontsize=16);
```


all Extractions concatenated
Pectin yield [g]
monoclonal Antibody / CBM
JIM5 JIM7 LM18 LM19 LM20 LM7 PAM1 LM16 LM8 2F4 INRA-RU2 INRA-RU1 LM5 LM6 LM13 LM12 LM10 LM11 LM15 LM21 LM22 (CBM3a (CBM30 BS-400-2 BS-400-3 LM2 JIM8 JIM13 JIM19 MAC207
0.092 0 56 0 0 41 0 0 0 0 0 0 15 7 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.126 0 63 0 0 40 0 0 0 0 0 0 9 5 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.139 0 63 0 0 47 0 0 0 0 0 0 11 6 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.148 0 62 0 0 46 0 0 0 0 0 0 10 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.15 0 71 0 0 47 0 0 0 0 0 0 11 5 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.158 0 67 0 0 46 0 0 0 0 0 0 9 6 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.173 0 63 0 0 49 0 0 0 0 0 0 9 5 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.177 0 61 0 0 47 0 0 0 0 0 0 7 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.183 0 69 0 0 45 0 0 0 0 0 0 8 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.203 0 71 0 0 47 0 0 0 0 0 0 8 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.21 0 66 0 0 46 0 0 0 0 0 0 5 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.23 0 68 0 0 49 0 0 0 0 0 0 6 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.231 0 70 0 0 48 0 0 0 0 0 0 9 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.234 0 75 0 0 49 0 0 0 0 0 0 7 0 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.234 0 76 0 0 51 0 0 0 0 0 0 6 0 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.23 0 76 0 0 52 0 0 0 0 0 0 6 0 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.252 0 81 0 0 49 0 0 0 0 0 0 8 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 0
0.256 0 73 0 0 47 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.266 0 82 0 0 51 0 0 0 0 0 0 5 0 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.276 0 73 0 0 55 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.289 0 74 0 0 42 0 0 0 0 0 0 5 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.275 0 81 0 0 51 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.288 0 77 0 0 49 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.09 0 53 0 0 37 0 0 0 0 0 0 14 6 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.121 0 63 0 0 45 0 0 0 0 0 0 17 10 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.151 0 64 0 0 46 0 0 0 0 0 0 13 7 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.218 0 75 0 0 53 0 0 0 0 0 0 12 7 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.249 0 70 0 0 53 0 0 0 0 0 0 10 0 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.297 0 60 0 0 45 0 0 0 0 0 0 6 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.307 0 73 0 0 48 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.108 18 39 0 0 34 0 0 0 0 0 0 7 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.191 29 81 0 0 72 0 0 0 0 0 18 39 33 26 0 0 0 0 0 0 0 0 0 0 0 0 0 10 0 0
0.199 22 66 0 0 58 0 0 0 0 0 12 28 22 19 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.256 28 78 0 0 67 0 0 0 0 0 26 41 35 22 0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0
0.29 25 76 0 0 66 0 0 0 0 0 29 40 37 23 0 0 0 6 0 0 5 0 0 0 0 0 0 7 0 0
0.312 23 55 0 0 48 0 0 0 0 0 26 30 26 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.326 24 69 0 0 63 0 0 0 0 0 35 43 39 19 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 0

### Perform Leave-One-Out Cross Validation for parameter tuning

To find the number of latent variables (LV) we perform LOO-CV. It is recommended to perform 5-fold CV to avoid overfitting. However, in this example the number of samples are quite limited, hence the choice for LOO-CV.

[7]:
```
from mbpls.mbpls import MBPLS
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import mean_squared_error

MSEs = []
for lv in range(15):
    mbpls = MBPLS(n_components=lv+1)
    prediction = cross_val_predict(mbpls, extractions_all, yield_all,␣
↪cv=len(extractions_all))
    prediction = pd.DataFrame(prediction)
    MSEs.append(mean_squared_error(prediction, yield_all))

plt.plot(np.arange(1,16), MSEs)
plt.xlabel('number of LVs', fontsize=16)
plt.ylabel('LOO-CV MSE', fontsize =16)
plt.title('Find the right number of LVs', fontsize =18);
```



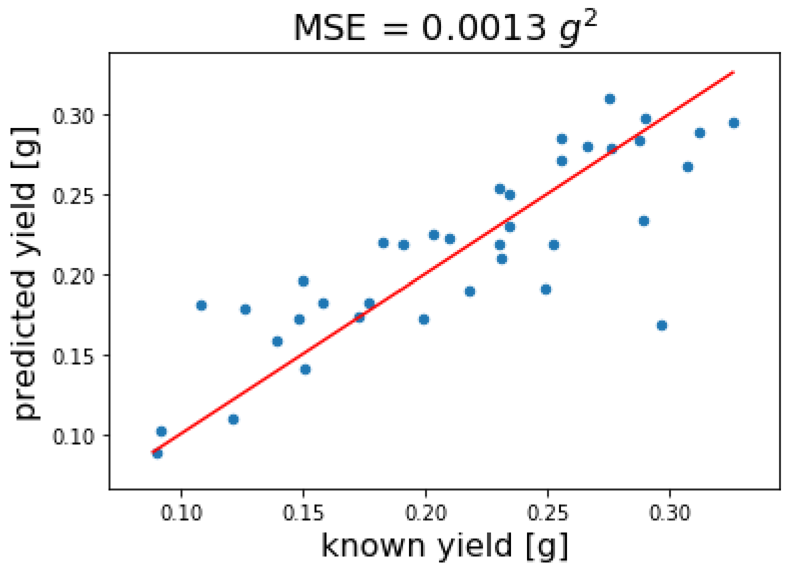
Finally, we show the calibration for a model with 5 LVs. The actual MSE minimum indicates 7 LVs. However, we want to prevent from overfitting and choose only 5 LVs (increased bias).

[8]:
```
mbpls = MBPLS(n_components=5)
prediction = cross_val_predict(mbpls, extractions_all, yield_all,
                               cv=len(extractions_all))
prediction = pd.DataFrame(prediction)
yield_all = pd.DataFrame(np.array(yield_all))
prediction = pd.concat((prediction, yield_all), axis=1)
prediction.columns=['predicted yield [g]', 'known yield [g]']
prediction.plot.scatter(x='known yield [g]', y='predicted yield [g]')
plt.plot([prediction.min().min(), prediction.max().max()],
         [prediction.min().min(), prediction.max().max()], color='red')
plt.ylabel(prediction.columns[0], fontsize=16)
plt.xlabel(prediction.columns[1], fontsize=16)
```

(continues on next page)

(continued from previous page)

```
plt.title('MSE = {:.4f} $g^2$'.format(mean_squared_error(prediction['known yield [g]
↪'],
                                                        prediction['predicted yield [g]'])),␣
↪fontsize=18);
```

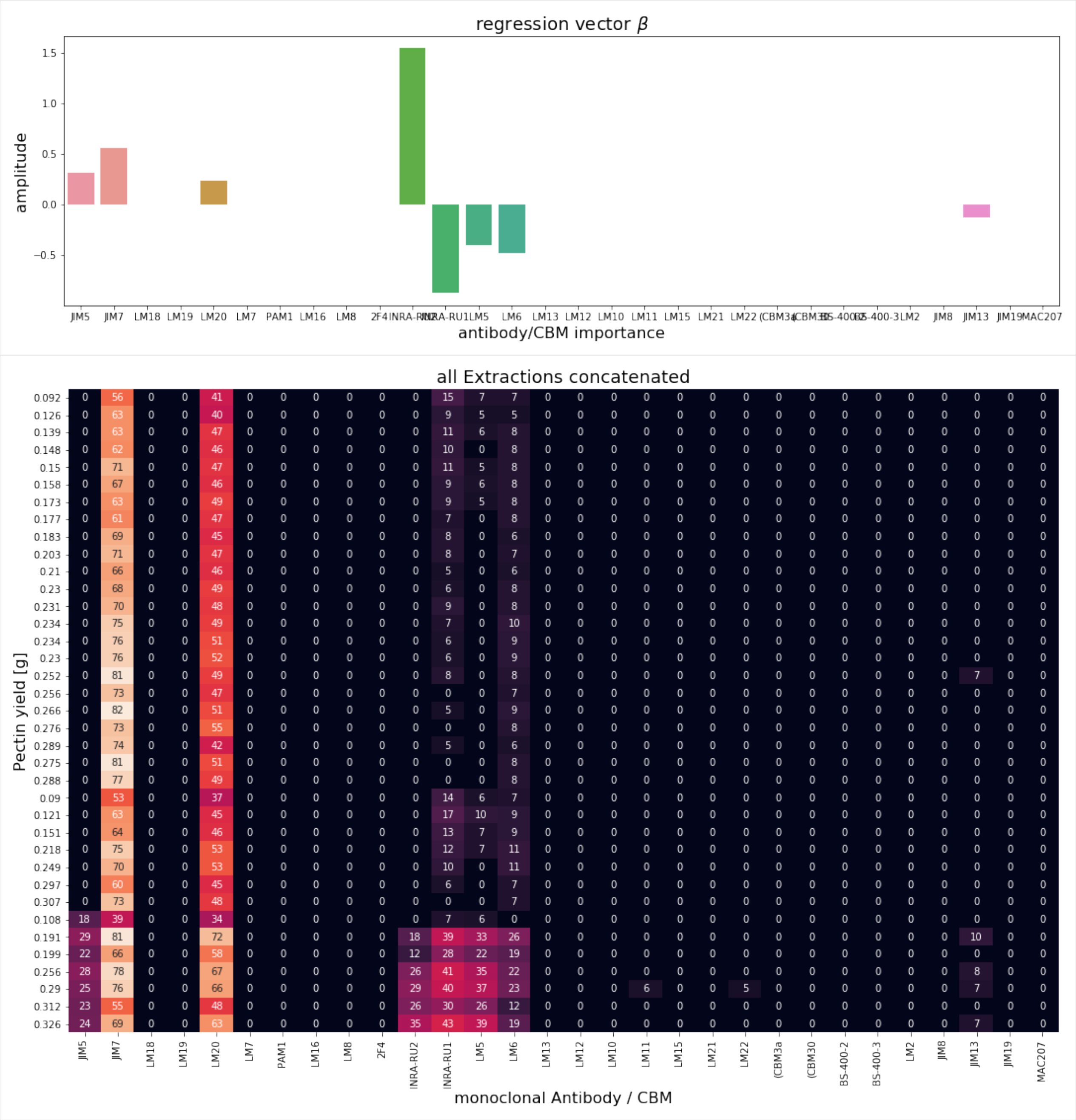


The calibration above indicates that carbohydrate microarrays can be used to predict pectin yield. The mean square error (MSE) is 0.0013 $g$

### Let's look at the regression vector $\beta$

To calculate the regression vector $\beta$ we first need to calculate a global model using our 5 LVs (we found these in the cross validation section).

[9]:
```
import seaborn as sns
mbpls = MBPLS(n_components=5)
mbpls.fit_transform(extractions_all, yield_all)
plt.figure(figsize=(18,5))
sns.barplot(x=extraction1.columns, y=mbpls.beta_[:,0])
plt.title('regression vector $\\beta$', fontsize=18)
plt.xlabel('antibody/CBM importance', fontsize=16)
plt.ylabel('amplitude', fontsize=16)

plt.figure(figsize=(18,12))
sns.heatmap(extractions_all, annot=True, cbar=False)
plt.title('all Extractions concatenated', fontsize=18)
plt.xlabel('monoclonal Antibody / CBM', fontsize=16)
plt.ylabel('Pectin yield [g]', fontsize=16);
```



| Pectin yield [g] | JIM5 | JIM7 | LM18 | LM19 | LM20 | LM7 | PAM1 | LM16 | LM8 | 2F4 | INRA-RU2 | INRA-RU1 | LM5 | LM6 | LM13 | LM12 | LM10 | LM11 | LM15 | LM21 | LM22 | (CBM3a | (CBM30 | BS-400-2 | BS-400-3 | LM2 | JIM8 | JIM13 | JIM19 | MAC207 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.092 | 0 | 56 | 0 | 0 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 15 | 7 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.126 | 0 | 63 | 0 | 0 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 5 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.139 | 0 | 63 | 0 | 0 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 11 | 6 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.148 | 0 | 62 | 0 | 0 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.15 | 0 | 71 | 0 | 0 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 11 | 5 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.158 | 0 | 67 | 0 | 0 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 6 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.173 | 0 | 63 | 0 | 0 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 5 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.177 | 0 | 61 | 0 | 0 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.183 | 0 | 69 | 0 | 0 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.203 | 0 | 71 | 0 | 0 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.21 | 0 | 66 | 0 | 0 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.23 | 0 | 68 | 0 | 0 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.231 | 0 | 70 | 0 | 0 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.234 | 0 | 75 | 0 | 0 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.234 | 0 | 76 | 0 | 0 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.23 | 0 | 76 | 0 | 0 | 52 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.252 | 0 | 81 | 0 | 0 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 |
| 0.256 | 0 | 73 | 0 | 0 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.266 | 0 | 82 | 0 | 0 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.276 | 0 | 73 | 0 | 0 | 55 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.289 | 0 | 74 | 0 | 0 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.275 | 0 | 81 | 0 | 0 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.288 | 0 | 77 | 0 | 0 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.09 | 0 | 53 | 0 | 0 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 14 | 6 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.121 | 0 | 63 | 0 | 0 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 17 | 10 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.151 | 0 | 64 | 0 | 0 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 13 | 7 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.218 | 0 | 75 | 0 | 0 | 53 | 0 | 0 | 0 | 0 | 0 | 0 | 12 | 7 | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.249 | 0 | 70 | 0 | 0 | 53 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.297 | 0 | 60 | 0 | 0 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.307 | 0 | 73 | 0 | 0 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.108 | 18 | 39 | 0 | 0 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.191 | 29 | 81 | 0 | 0 | 72 | 0 | 0 | 0 | 0 | 0 | 18 | 39 | 33 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 0 |
| 0.199 | 22 | 66 | 0 | 0 | 58 | 0 | 0 | 0 | 0 | 0 | 12 | 28 | 22 | 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.256 | 28 | 78 | 0 | 0 | 67 | 0 | 0 | 0 | 0 | 0 | 26 | 41 | 35 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 0 |
| 0.29 | 25 | 76 | 0 | 0 | 66 | 0 | 0 | 0 | 0 | 0 | 29 | 40 | 37 | 23 | 0 | 0 | 0 | 6 | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 |
| 0.312 | 23 | 55 | 0 | 0 | 48 | 0 | 0 | 0 | 0 | 0 | 26 | 30 | 26 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.326 | 24 | 69 | 0 | 0 | 63 | 0 | 0 | 0 | 0 | 0 | 35 | 43 | 39 | 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 |

From the regression vector $\beta$ (see above) one can understand which antibodies and/or CBMs are important for prediction of prectin yield. Additionally, the signs of the individual coefficents indicate positive or negative correlations, respectively.

## References

[1] Baum, Andreas, et al. "Prediction of pectin yield and quality by FTIR and carbohydrate microarray analysis." Food and Bioprocess Technology 10.1 (2017): 143-154.

### PLS for prediction of pectin yield from FTIR spectroscopic measurement of crude extracts

This Jupyter notebook illustrates the use of PLS to establish prediction models for the pectin yield from Fourier Transform Infrared (FTIR) spectra measured on crude extracts at different time points and conditions. The data was obtained from [1] and has been acquired using a FTIR instrument (FOSS FT2), which is capable of measuring light absorption in the mid infrared spectral region. Depending on the extraction condition and extraction time point the obtained spectra resemble fingerprints of the extract and its overall chemical complexity. Typically, such spectroscopic data contains measurements at many wavenumbers (= features $p$). PLS is often used for these types of data because it can handle high dimensional datasets with highly correlated features (as it is the case in this example).

We aim to establish a PLS prediction model such that:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{E}$$

The data matrix **X** indicates the shape $n \times p$, where $n$ resembles the number of samples and $p$ the number of features. Once we have found the regression vector $\boldsymbol{\beta}$ we can apply it to newly measured samples of similar type to predict the respective pectin yield of these samples. The variation in **Y** which is not explained by the model resembles the residuals **E**

#### Import dependencies and load data

Three datasets are imported for this example. Extraction 1 and 2 were carried out enzymatically, while extraction 3 was carried out using an acid extraction procedure. The three datasets are merged to be analyzed collectively (named *ftir*). The data in *yield_all* contains the final pectin yields (**y**). If you encounter errors in this section you might need to install/upgrade some of the Python packages stated below.

[2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from mbpls.data.get_data import load_FTIR_Data
from scipy.io import loadmat
```

[3]:

```
data = load_FTIR_Data()
```

```
Following dataset were loaded as Pandas Dataframes:
 dict_keys([’ftir3’, ’ftir1’, ’ftir2’])
```

[4]:

```
ftir1 = data['ftir1']
ftir2 = data['ftir2']
ftir3 = data['ftir3']
ftir = pd.concat((ftir1, ftir2, ftir3))
yield_all = np.array(ftir.index)
wavenumbers = ftir1.columns
```

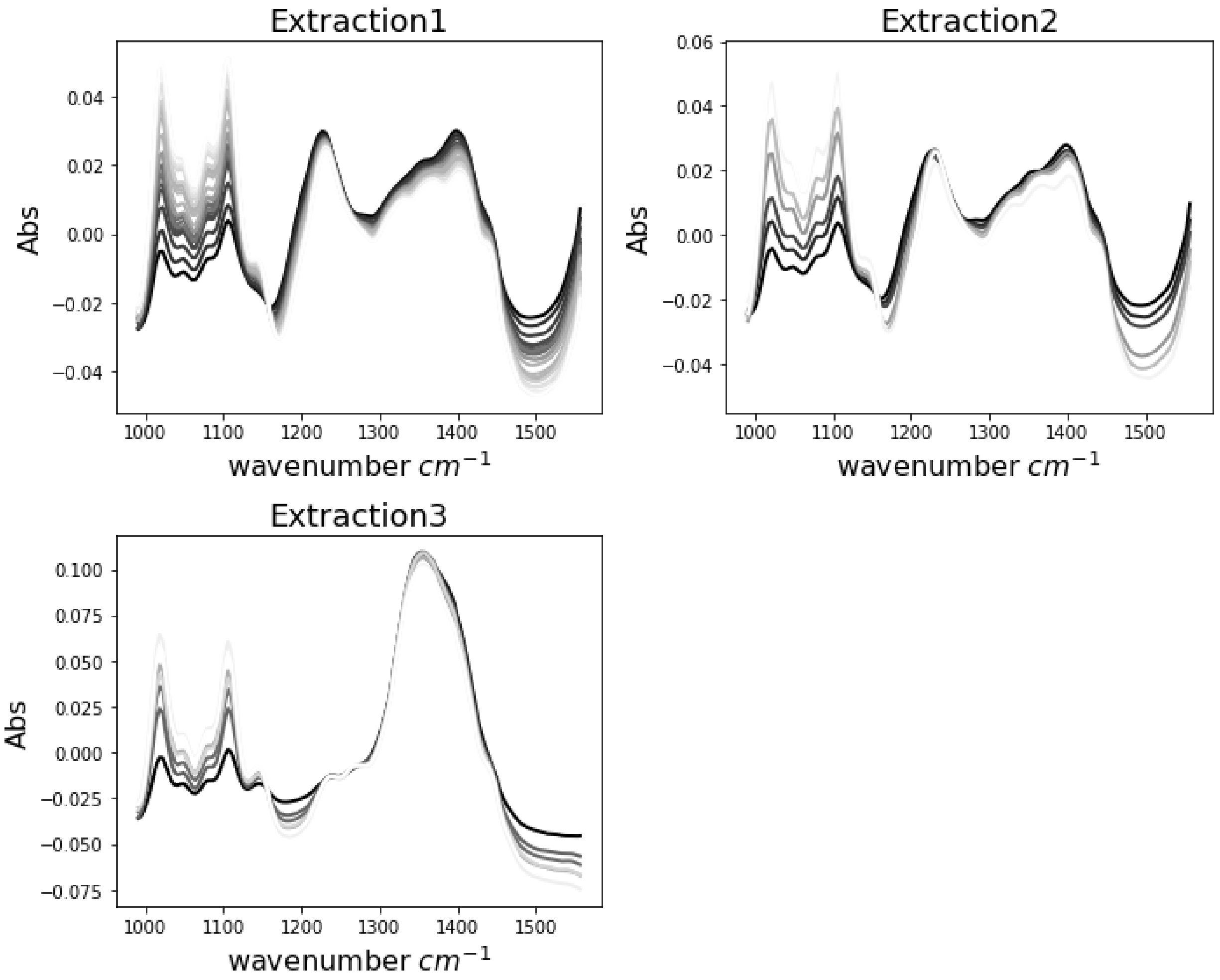
#### Look at the data

First we want to look at our data. We have created a function which helps us to do so. Each of the three data sets contains several time points at which the measurement was repeated during the extraction. The spectra are colored in greyscale, resembling the final pectin yield (dark - lowest, white - highest).

One can observe that extractions 1 and 2 appear to have similar shape, while extraction 3 indicates a very different characteristics. Remember that extraction 3 was carried out using acid. All extractions show changes in the spectra which seem to be related to final pectin yield. These spectral changes are especially abundant in the wavenumber range between 1000 and 1150 $cm^{-1}$.

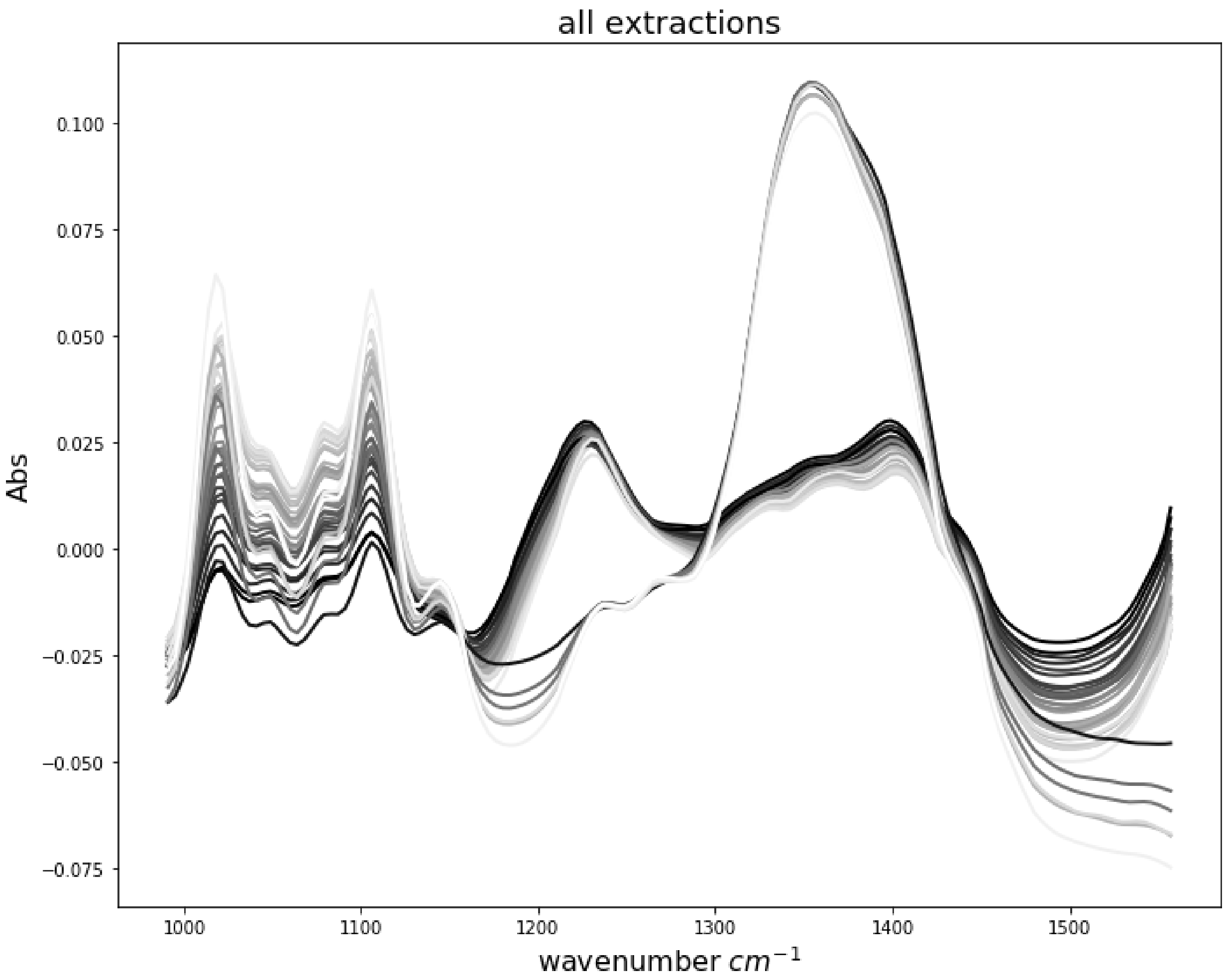
[5]:

```
def plot_spectra(spectra, ax, name):
    pectin_yield = np.array(spectra.index)
    color_code = (pectin_yield - pectin_yield.min())
    color_code = color_code / color_code.max()
    color_code = color_code[:]
    for spectrum, color in zip(np.array(spectra), color_code):
        ax.plot(wavenumbers, spectrum, color=(color, color, color), linewidth=2)
        ax.set_title(name, fontsize=18)
        ax.set_xlabel('wavenumber $cm^{-1}$', fontsize=16)
        ax.set_ylabel('Abs', fontsize=16)
        plt.tight_layout()

fig, ax = plt.subplots(ncols=2, nrows=2, figsize=(10,8))
plot_spectra(ftir1, ax[0][0], 'Extraction1')
plot_spectra(ftir2, ax[0][1], 'Extraction2')
plot_spectra(ftir3, ax[1][0], 'Extraction3')
ax[1][1].axis('off');
```



When we look at all spectra plotted together we obtain the following situation. It is difficult to identify which spectral wavenumbers are related to higher pectin yields. This is due to the fact, that both extraction patterns, enzymatic and acidic, appear superimposed. In the following, we will use PLS regression to establish prediction models using this combined data set.

[6]:
```
yield_all = np.array(ftir.index)
fig, ax = plt.subplots(figsize=(10,8))
plot_spectra(ftir, ax, 'all extractions')
```



### Find number of latent variables using Leave-One-Out Cross Validation

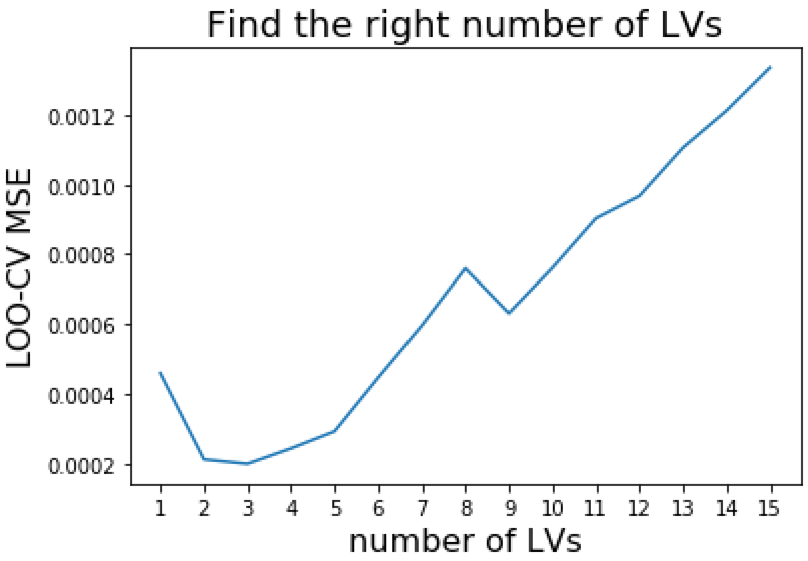
To find the right number of latent variables (LV) we perform cross validation. Scikit-learn makes it easy for us using the cross_val_predict function. We simply re-model the data using different numbers of latent variables and plot the result of the resulting Mean Square Errors (MSE). We pick the number of latent variables where the MSE is minimal to avoid overfitting and to induce as much complexity into our model as to obtain most accurate results.

[7]:
```
from mbpls.mbpls import MBPLS
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import mean_squared_error

MSEs = []
for lv in range(15):
    mbpls = MBPLS(n_components=lv+1)
    prediction = cross_val_predict(mbpls, ftir, yield_all, cv=len(ftir))
    prediction = pd.DataFrame(prediction)
    MSEs.append(mean_squared_error(prediction, yield_all))
```

(continues on next page)

(continued from previous page)

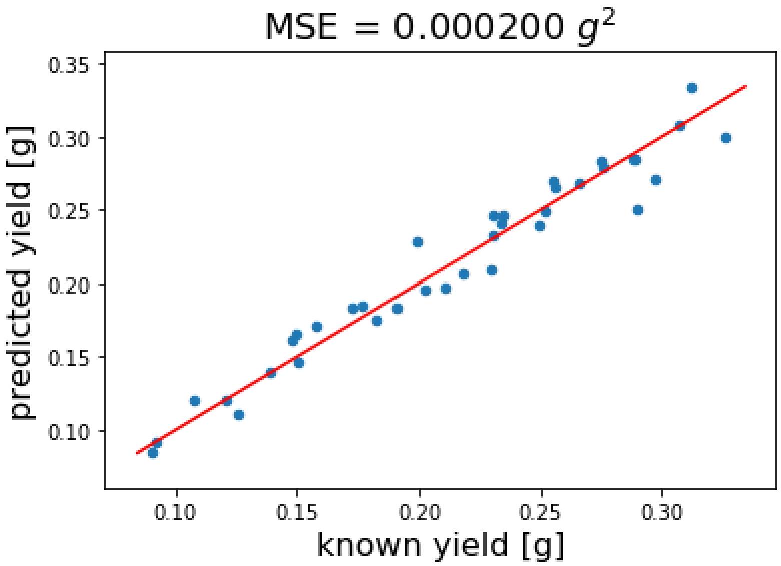
```
plt.plot(np.arange(1,16), MSEs)
plt.xlabel('number of LVs', fontsize=16)
plt.xticks(np.arange(1,16), np.arange(1,16))
plt.ylabel('LOO-CV MSE', fontsize=16)
plt.title('Find the right number of LVs', fontsize=18);
```



### Plot resulting calibration and evaluate model

Once we have found the right number of latent variables we re-use this number to calculate our final model. Before we do this using all data (no CV) we want to look at the predicted versus known pectin yield plot below. As you can see our model predicts pectin yield quite well with a mean squared error of 0.0002 $g^2$.

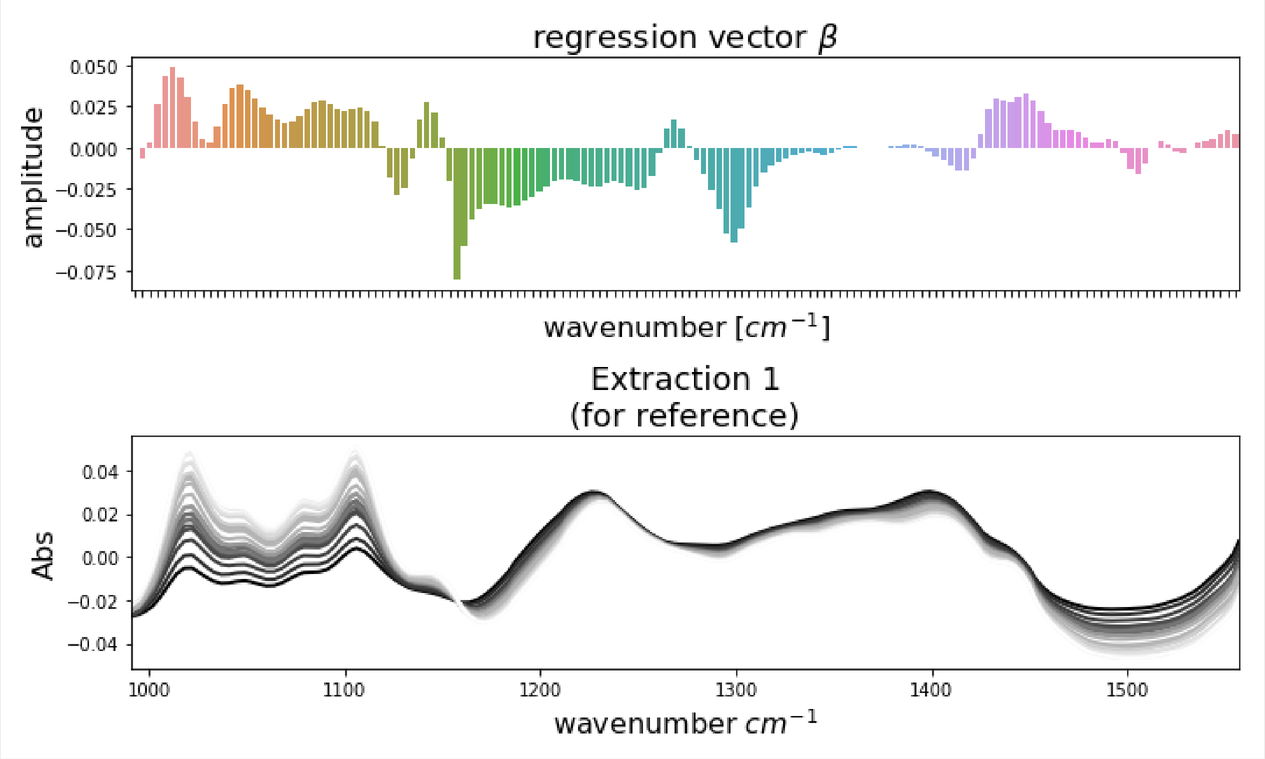
```
[8]: mbpls = MBPLS(n_components=3, method='NIPALS')
prediction = cross_val_predict(mbpls, ftir, yield_all, cv=len(ftir))
prediction = pd.DataFrame(prediction)
yield_all = pd.DataFrame(np.array(yield_all))
prediction = pd.concat((prediction, yield_all), axis=1)
prediction.columns=['predicted yield [g]', 'known yield [g]']
prediction.plot.scatter(x='known yield [g]', y='predicted yield [g]')
plt.plot([prediction.min().min(), prediction.max().max()],
         [prediction.min().min(), prediction.max().max()], color='red')
plt.ylabel(prediction.columns[0], fontsize=16)
plt.xlabel(prediction.columns[1], fontsize=16)
plt.title('MSE = {:.6f} $g^2$'.format(mean_squared_error(prediction['known yield [g]
↪'],
                                   prediction['predicted yield [g]'])),fontsize=18);
```



### Calculate final model and look at regression vector $\beta$

Now we have validated our model and we are ready to calculate the final model using all available samples. After doing so we can plot the regression vector $\boldsymbol{\beta}$ and use it to pedict the pectin yield of future samples of similar type. To compare $\boldsymbol{\beta}$ to the actual spectral patterns in the data, we have plotted it below together with the extraction 1 samples. It is easy to conclude which wavenumbers have positiv impact on pectin yield and vice versa. Of course, one could also compare $\boldsymbol{\beta}$ to the spectra of extractions 2 and 3. This can be done by replacing `ftir1` with `ftir2` or `ftir3` in the code below. Note, that the regression vector $\boldsymbol{\beta}$ is applied to standardized data (standardization is enabled by default).

[9]:
```
import seaborn as sns
mbpls = MBPLS(n_components=5)
mbpls.fit_transform(ftir, yield_all)
fig, ax = plt.subplots(nrows=2, figsize=(10,6))
sns.barplot(x=ftir.columns, y=mbpls.beta_[:,0], ax=ax[0])
ax[0].set_title('regression vector $\\beta$', fontsize=18)
ax[0].set_xlabel('wavenumber $[cm^{-1}]$', fontsize=16)
ax[0].set_ylabel('amplitude', fontsize=16)
ax[0].axes.xaxis.set_ticklabels([])
plot_spectra(ftir1, ax[1], 'Extraction 1\n(for reference)')
ax[1].set_xlim([wavenumbers.min(), wavenumbers.max()]);
```



### References

[1] Baum, Andreas, et al. "Prediction of pectin yield and quality by FTIR and carbohydrate microarray analysis." Food and Bioprocess Technology 10.1 (2017): 143-154.

## 2.2.3 API Reference

### Multi-Block Partial Least Squares (MB-PLS) for Python

The mbpls package contains three multi-block capable algorithms, i.e. KERNEL, NIPALS and UNIPALS, as well as SIMPLS for fast predictions.

The aim of the package is to provide a unified interface and easy access to these algorithms.

### mbpls.mbpls module

### Module contents

**class** mbpls.mbpls.**MBPLS**(*n_components=2*, *full_svd=False*, *method='NIPALS'*, *standardize=True*, *max_tol=1e-14*, *calc_all=True*, *sparse_data=False*)

- **PLS1**: Predict a response vector $y$ from a single multivariate data block $X$
- **PLS2**: Predict a response matrix $Y$ from a single multivariate data block $X$
- **MBPLS**: Predict a response vector/matrix $Y$ from multiple data blocks $X_1, X_2, ..., X_i$

for detailed information check [ref]

**method** [string (default 'NIPALS')] The method being used to derive the model attributes, possible are 'UNIPALS', 'NIPALS', 'SIMPLS' and 'KERNEL'

**n_components** [int] Number ($k$) of Latent Variables (LV)

**standardize** [bool (default True)] Standardizing the data

**full_svd** [bool (default True)] Using full singular value decomposition when performing SVD method. Set to 'False' when using very large quadratic matrices $X$.

**max_tol** [non-negative float (default 1e-14)] Maximum tolerance allowed when using the iterative NIPALS algorithm

**calc_all** [bool (default True)] Calculate all internal attributes for the used method. Some methods do not need to calculate all attributes, i.e. scores, weights etc., to obtain the regression coefficients used for prediction. Setting this parameter to false will omit these calculations for efficiency and speed.

**sparse_data** [bool (default False)] NIPALS is the only algorithm that can handle sparse data using the method of H. Martens and Martens (2001) (p. 381). If this parameter is set to 'True', the method will be forced to NIPALS and sparse data is allowed. Without setting this parameter to 'True', sparse data will not be accepted.

**X-side**

**Ts_** : array, super scores $[n, k]$

**T_** : list, block scores $[i][n, k]$

**W_** : list, block weights $[i][p_i, k]$

**A_** : array, block importances/super weights $[i, k]$

**A_corrected_** : array, normalized block importances $A_{corr,ik} = A_{ik} \cdot (1 - \frac{p_i}{p})$

**P_** : list, block loadings $[i][p_i, k]$

**R_** : array, x_rotations $R = W(P^T W)^{-1}$

**explained_var_x_** : list, explained variance in $X$ per LV $[k]$

**explained_var_xblocks_** : array, explained variance in each block $X_i$ $[i, k]$

**beta_** : array, regression vector $\beta$ $[p, q]$

**Y-side**

**U_** : array, scoresInitialize $[n, k]$

**V_** : array, loadings $[q, k]$

**explained_var_y_** : list, explained variance in $Y$ $[k]$

## Notes

According to literature one distinguishes between PLS1 [ref], PLS2 [ref] and MBPLS [ref]. Common goal is to find loading vectors $p$ and $v$ which project the data to latent variable scores $ts$ and $u$ indicating maximal covariance. Subsequently, the explained variance is deflated and further LVs can be extracted. Deflation for the $k$-th LV is obtained as:

$$X_{k+1} = X_k - t_k p_k^T$$

**PLS1**: Matrices are computed such that:

$$
\begin{aligned}
X &= T_s P^T + E_X \\
y &= X\beta + e
\end{aligned}
$$

**PLS2**: Matrices are computed such that:

$$
\begin{aligned}
X &= T_s P^T + E_X \\
Y &= UV^T + E_Y \\
Y &= X\beta + E
\end{aligned}
$$

**MBPLS**: In addition, MBPLS provides a measure for how important ($a_{ik}$) each block $X_i$ is for prediction of $Y$ in the $k$-th LV. Matrices are computed such that:

$$
\begin{aligned}
X &= [X_1|X_2|...|X_i] \\
X_i &= T_s P_i^T + E_i \\
Y &= UV^T + E_Y \\
Y &= X\beta + E
\end{aligned}
$$

using the following calculation:

$X_k = X$

for k in K:

$$
\begin{aligned}
w_k &= \text{first eigenvector of } X_k^T Y Y^T X_k, ||w_k||_2 = 1 \\
w_k &= [w_{1k}|w_{2k}|...|w_{ik}] \\
a_{ik} &= ||w_{ik}||_2^2 \\
t_{ik} &= \frac{X_i w_{ik}}{||w_{ik}||_2} \\
t_{sk} &= \sum a_{ik} * t_{ik} \\
v_k &= \frac{Y^T t_{sk}}{t_{sk}^T t_{sk}} \\
u_k &= Y v_k \\
u_k &= \frac{u_k}{||u_k||_2} \\
p_k &= \frac{X^T t_{sk}}{t_{sk}^T t_{sk}}, p_k = [p_{1k}|p_{2k}|...|p_{ik}] \\
X_{k+1} &= X_k - t_{sk} p_k
\end{aligned}
$$

End loop

$P = [p_1|p_2|...|p_K]$

$T_s = [t_{s1}|t_{s2}|...|t_{sK}]$

$U = [u_1|u_2|...|u_K]$

$V = [v_1|v_2|...|v_K]$

$W = [w_1|w_2|...|w_k]$

$R = W(P^T W)^{-1}$

$\beta = RV^T$

### Examples

Quick Start: Two random data blocks $X_1$ and $X_2$ and a random reference vector $y$ for predictive modeling.

```
import numpy as np
from mbpls.mbpls import MBPLS

mbpls = MBPLS(n_components=4)
x1 = np.random.rand(20,300)
x2 = np.random.rand(20,450)

y = np.random.rand(20,1)

mbpls.fit([x1, x2],y)
mbpls.plot(num_components=4)

y_pred = mbpls.predict([x1, x2])
```

More elaborate examples can be found at https://github.com/DTUComputeStatisticsAndDataAnalysis/MBPLS/tree/master/examples

**check_sparsity_level** (*data*)

**explained_variance_score** (*X*, *Y*)

**fit** (*X*, *Y*)

Fit model to given data

**Parameters**

- **X** (*list*) – of all xblocks x1, x2, ..., xn. Rows are observations, columns are features/variables
- **Y** (*array*) – 1-dim or 2-dim array of reference values

**fit_predict** (*X*, *Y*, ***fit_params*)

Fit to data, then predict it.

**fit_transform** (*X*, *y=None*, ***fit_params*)

Fit the model and then, then transform the given data to lower dimensions.

**plot** (*num_components=2*)

Function that prints the fitted values of the instance. INPUT: num_components: Int or list

Int: The number of components that will be plotted, starting with the first component list: Indices or range of the components that should be plotted

**predict** (*X*)

Predict y based on the fitted model

**Parameters X** (*list*) – of all xblocks x1, x2, ..., xn. Rows are observations, columns are features/variables

**Returns**

- **y_hat** (*np.array*)
- *Predictions made based on trained model and supplied X*

**r2_score** (*X*, *Y*)

**transform** (*X*, *Y=None*, *return_block_scores=False*)

Obtain scores based on the fitted model

Parameters

**X** [list] of arrays containing all xblocks x1, x2, . . . , xn. Rows are observations, columns are features/variables

**(optional) Y** [array] 1-dim or 2-dim array of reference values

**return_block_scores: bool (default False)** Returning block scores **T_** when transforming the data

**Returns**

- **Super_scores** (*np.array*)
- **Block_scores** (*list*)
- *List of np.arrays containing the block scores*
- **Y_scores** (*np.array (optional)*)
- *Y-scores, if y was given*

### mbpls.data module

### Module contents

The *mbpls.data* module contains methods to load data real world datasets and to create artificial data than can be used to test the mbpls methods.

mbpls.data.**data_path**()

mbpls.data.**orthogonal_data**(*num_of_samples=11*, *params_block_one=4*, *params_block_two=4*, *params_block_three=4*, *num_of_variables_main_lin_comb=0*, *num_of_batches=1*, *random_state=None*)

This function creates a dataset with three X-blocks, which are completely orthogonal amongst each other and one Y-block, that has two response variables, which are a linear combination of the variables defined for the three blocks.

num_of_samples: Amount of samples for the dataset params_block_one: Number of variables in the first block params_block_two: Number of variables in the second block params_block_three: Number of variables in the third block num_of_variables_main_lin_comb: Number of variables that are randon linear combinations of each variable (Multi-Colliniearity) num_of_batches: Number of batches for each block (third dimension)

X_1 = First X-block - Dimensionality ( num_of_samples, params_block_one*(num_of_variables_main_lin_comb+1), num_of_batches) X_2 = Second X-block - Dimensionality ( num_of_samples, params_block_two*(num_of_variables_main_lin_comb+1), num_of_batches) X_3 = Third X-block - Dimensionality ( num_of_samples, params_block_three*(num_of_variables_main_lin_comb+1), num_of_batches) Y = Y-block - Dimensionality (num_of_samples, 2, num_of_batches)

mbpls.data.**load_CarbohydrateMicroarrays_Data**()

mbpls.data.**load_FTIR_Data**()

mbpls.data.**load_Intro_Data**()

### mbpls.tests module

### Module contents

The `mbpls.tests` module contains an installation test script, which contains a range of tests created for the pytest package that ensure the correct installation of the mbpls package and subsequently the recreation of predefined results for all methods. This is especially designed to verify the validity of results after making changes to the source code of the implemented algorithms.

### mbpls.tests.test_Installation module

CHAPTER

THREE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

# INDEX