Adadmire installation and users guide for windows

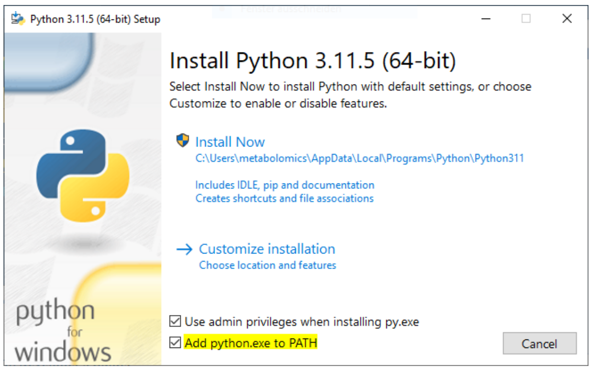
General remarks

Adadmire or short Admire tries to find a generalized model that establishes relationships between continuous and discrete data. After training Admire predicts a certain value e.g. metabolite concentration based on all other values of this sample, therefore, Admire is only as good as the used training data. Furthermore, if you have a single specimen where for example the citrate concentration is extremely high, while all other concentrations of this specimen are within their normal ranges observed across the whole data set Admire will detect this citrate value as an outlier and will try to correct it. However, this outlier may still be correct for example due to a specific treatment of the corresponding proband. Therefore, when Admire detects an outlier it is a good idea to go back to your original data for verification.

This guide will help you test Admire on your data, even if you have little to no coding experience. The whole process would take around 10 minutes, given you have a good internet connection and a PC with average specs.

Installing **Python**:

If Python is not already installed, download it from https://www.python.org/downloads/
On the first setup window (see screenshot), make sure to activate the “Add python.exe to PATH” checkbox below (Python 3.11.5 installation). This ensures that PowerShell will recognize the relevant command lines later on.
Python should be found on a path similar to:


Python 3.11.5 (64-bit) Setup

Install Python 3.11.5 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\metabolomics\AppData\Local\Programs\Python\Python311

Includes IDLE, pip and documentation
Creates shortcuts and file associations

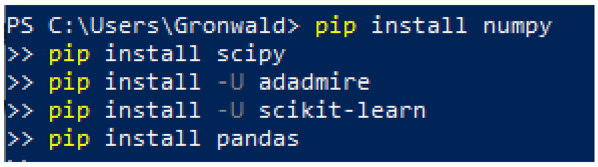
Customize installation
Choose location and features

☑ Use admin privileges when installing py.exe
☑ Add python.exe to PATH

C:\Users\Username\AppData\Local\Programs\Python\Python3xx

In **PowerShell**:

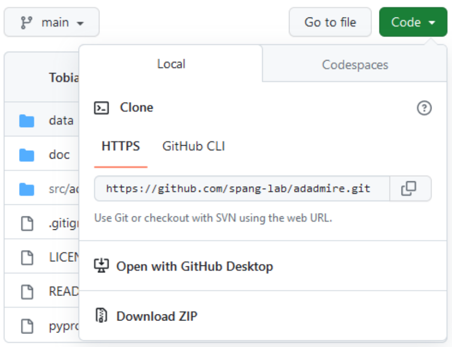
Installation of required packages for Python (In the search field in the Start menu, search for Windows PowerShell).
In Windows PowerShell paste the following code and press Enter:
pip install numpy # matrices
pip install -U adadmire # ADMIRE
pip install pandas # data reading and writing


```
PS C:\Users\Gronwald> pip install numpy
>> pip install scipy
>> pip install -U adadmire
>> pip install -U scikit-learn
>> pip install pandas
```

Admire will be installed in
C:\Users\Username\AppData\Local\Programs\Python\Python311\Lib\site-packages\adadmire
This takes around one minute.
Note, you have to install these packages only once.

Downloading the Test Data from **GitHub**:

Data available at https://github.com/spang-lab/adadmire
Click on the green “Code” button and choose “Download ZIP”
Unzip the data to an easily reachable path (for convenience purpose only) such as ('**D:/adadmire-main/**') for instance.


https://github.com/spang-lab/adadmire.git

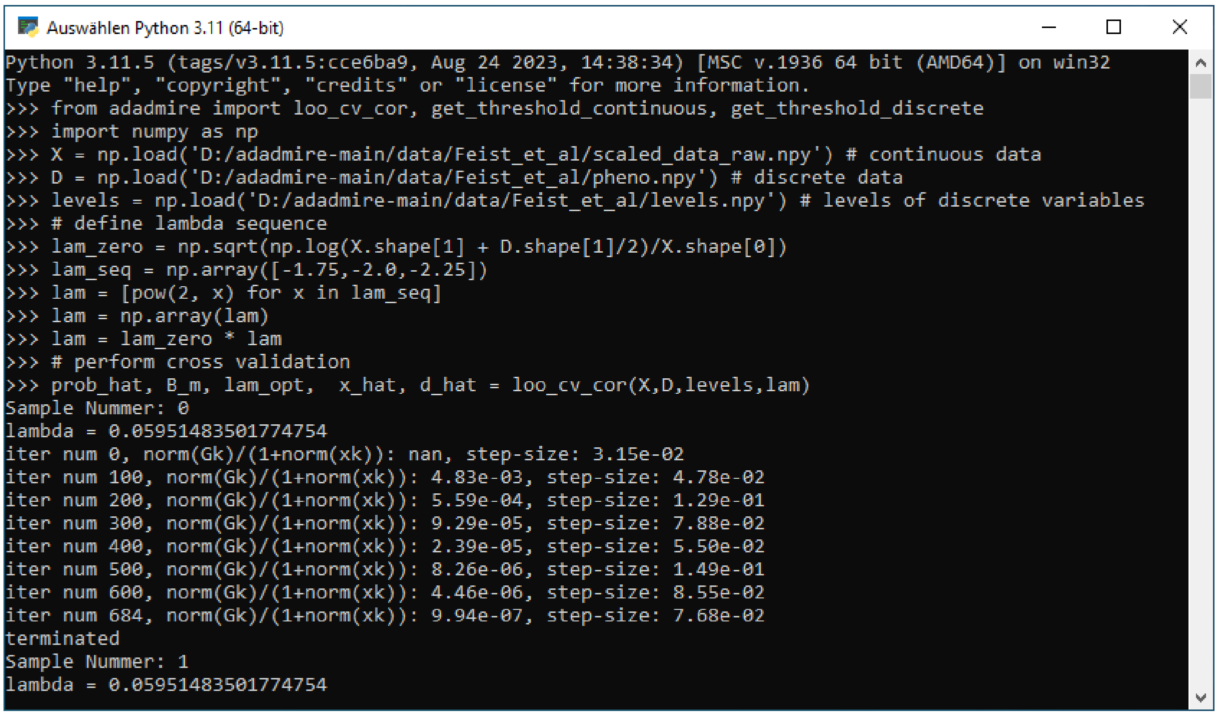
Use Git or checkout with SVN using the web URL.

In the test data three different test sets including required code are included:
Test set 1: standard application of ADMIRE

Test set 2: introduction of artificial anomalies to check the performance of ADMIRE
Test set 3: application of ADMIRE on a data set containing missing values.

Now in **Python run ADMIRE**:
We will use Test set 1 in this example
Start Python by typing python in a powershell. Alternatively you can also run Python from R-Studio
Alter the path of the input/output data files (shown in bold) in the following code, and paste it into Python:

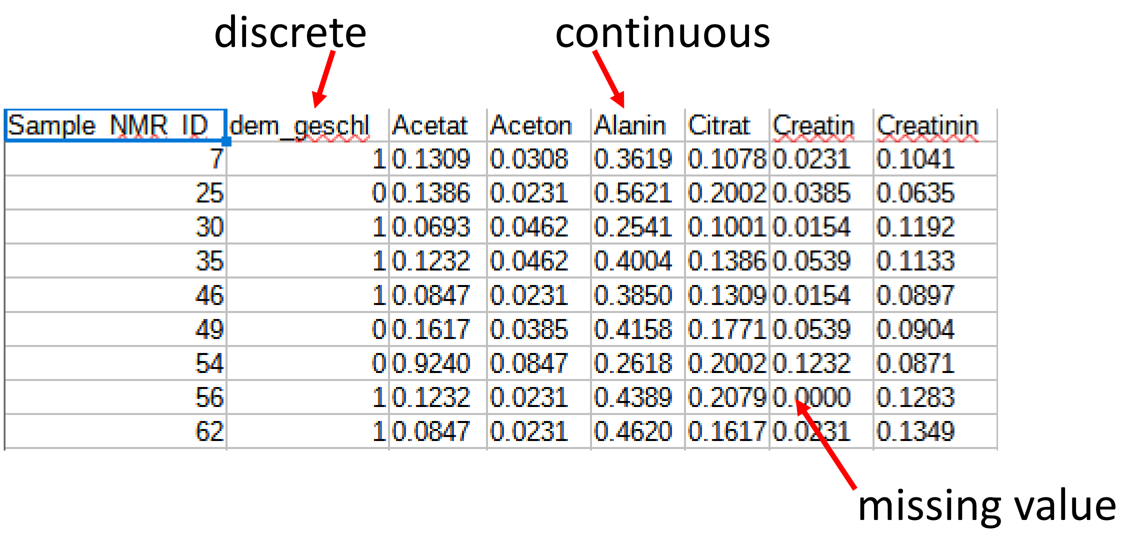
```
from adadmire import loo_cv_cor, get_threshold_continuous, get_threshold_discrete
import numpy as np
X = np.load('D:/adadmire-main/data/Feist_et_al/scaled_data_raw.npy') # continuous data
D = np.load('D:/adadmire-main/data/Feist_et_al/pheno.npy') # discrete data
levels = np.load('D:/adadmire-main/data/Feist_et_al/levels.npy') # levels of discrete variables
# define lambda sequence
lam_zero = np.sqrt(np.log(X.shape[1] + D.shape[1]/2)/X.shape[0])
lam_seq = np.array([-1.75,-2.0,-2.25])
lam = [pow(2, x) for x in lam_seq]
lam = np.array(lam)
lam = lam_zero * lam
# perform cross validation
prob_hat, B_m, lam_opt,  x_hat, d_hat = loo_cv_cor(X,D,levels,lam)
# determine continuous threshold
X_cor, threshold_cont, n_ano_cont,  position_cont = get_threshold_continuous(X, x_hat, B_m)
# returns: X corrected for detected anomalies, threshold, number of detected anomalies (n_ano_cont) and position
np.savetxt("D:/adadmire-main/X_cor.csv",X_cor,delimiter=";") # export X_cor as csv
print(n_ano_cont) # 46 detected continuous anomalies
n_ano_disc, threshold_cont, position_disc = get_threshold_discrete(D, levels, d_hat)
# returns:  number of detected anomalies (n_ano_disc), threshold and position
print(n_ano_disc)
# 0 detected discrete anomalies
print(position_cont) # shows positions of the anomalies
```



```
Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from adadmire import loo_cv_cor, get_threshold_continuous, get_threshold_discrete
>>> import numpy as np
>>> X = np.load('D:/adadmire-main/data/Feist_et_al/scaled_data_raw.npy') # continuous data
>>> D = np.load('D:/adadmire-main/data/Feist_et_al/pheno.npy') # discrete data
>>> levels = np.load('D:/adadmire-main/data/Feist_et_al/levels.npy') # levels of discrete variables
>>> # define lambda sequence
>>> lam_zero = np.sqrt(np.log(X.shape[1] + D.shape[1]/2)/X.shape[0])
>>> lam_seq = np.array([-1.75,-2.0,-2.25])
>>> lam = [pow(2, x) for x in lam_seq]
>>> lam = np.array(lam)
>>> lam = lam_zero * lam
>>> # perform cross validation
>>> prob_hat, B_m, lam_opt,  x_hat, d_hat = loo_cv_cor(X,D,levels,lam)
Sample Nummer: 0
lambda = 0.05951483501774754
iter num 0, norm(Gk)/(1+norm(xk)): nan, step-size: 3.15e-02
iter num 100, norm(Gk)/(1+norm(xk)): 4.83e-03, step-size: 4.78e-02
iter num 200, norm(Gk)/(1+norm(xk)): 5.59e-04, step-size: 1.29e-01
iter num 300, norm(Gk)/(1+norm(xk)): 9.29e-05, step-size: 7.88e-02
iter num 400, norm(Gk)/(1+norm(xk)): 2.39e-05, step-size: 5.50e-02
iter num 500, norm(Gk)/(1+norm(xk)): 8.26e-06, step-size: 1.49e-01
iter num 600, norm(Gk)/(1+norm(xk)): 4.46e-06, step-size: 8.55e-02
iter num 684, norm(Gk)/(1+norm(xk)): 9.94e-07, step-size: 7.68e-02
terminated
Sample Nummer: 1
lambda = 0.05951483501774754
```

<u>Applying adadmire (ADMIRE) on your **own Data**</u>

# General remarks
# In the following, we assume that continous and discrete data are combined in one .csv file (e.g. by
# saving .xlsx file to .csv) looking like this:


discrete
continuous
missing value

Sample NMR ID	dem_geschl	Acetat	Aceton	Alanin	Citrat	Creatin	Creatinin
7	1	0.1309	0.0308	0.3619	0.1078	0.0231	0.1041
25	0	0.1386	0.0231	0.5621	0.2002	0.0385	0.0635
30	1	0.0693	0.0462	0.2541	0.1001	0.0154	0.1192
35	1	0.1232	0.0462	0.4004	0.1386	0.0539	0.1133
46	1	0.0847	0.0231	0.3850	0.1309	0.0154	0.0897
49	0	0.1617	0.0385	0.4158	0.1771	0.0539	0.0904
54	0	0.9240	0.0847	0.2618	0.2002	0.1232	0.0871
56	1	0.1232	0.0231	0.4389	0.2079	0.0000	0.1283
62	1	0.0847	0.0231	0.4620	0.1617	0.0231	0.1349

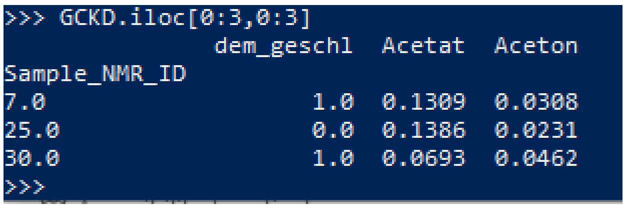
# Note it is also possible to keep continuous and discrete variables separated in two files.

# Start python
# From here everything in python, start Python by typing ‘python’, all comments marked by #
python

# Load required functions
from adadmire import loo_cv_cor, get_threshold_continuous, get_threshold_discrete, impute
import numpy as np

from sklearn import preprocessing #for preprocessing
import pandas as pd # to read.csv files

# Read data
GCKD = pd.read_csv('C:/Users/**Username**/own_test.csv', delimiter=',',dtype='float',index_col=0)
# Note, row and column numbers start with 0 in Python. Show data with e.g.: 'Name.iloc[0:3,0:3]
# Result looks like this:



```
>>> GCKD.iloc[0:3,0:3]
               dem_geschl  Acetat  Aceton
Sample_NMR_ID
7.0                   1.0  0.1309  0.0308
25.0                  0.0  0.1386  0.0231
30.0                  1.0  0.0693  0.0462
>>>
```

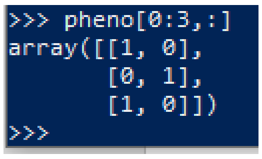
# Separate continuous and discrete data
# Get continuous data, note column zero now contains discrete data as content of column zero was
# shifted to row names
GCKD_con=np.copy(GCKD.iloc[:,1:GCKD.shape[1]]) # continuous data, omit discrete data here

# Replace zeros
# In this example missing data are represented by zeros, for ADMIRE these zeros have to be replaced
# by 'nan'
GCKD_con[GCKD_con == 0] = float("nan")
print(np.sum(np.isnan(GCKD_con))) # count nans

# Standardize
# ADMIRE requires standardized data (mean = 0 and variance =1).
GCKD_con_s = preprocessing.scale(GCKD_con)
# note it is now an np array access with e.g. 'GCKDs[0:3,0:3]'. Note row and col names are gone but
# can be added later.with 'np.nansum(GCKD_con_s, axis=0)' shows column sums should be all close #
to zero after standardization. With axis=1 you get sum of rows. np.nansum treats nans as zeros here.

# Discrete pheno data
# Here, sex (m/f) coded as 1 and 0, ADMIRE requires here array of 2 columns, one column codes
# whether a subject is male and the other whether a subject is female i.e. when we have in one
# column a 1 we need a 0 in the other one. For each group one column required.

```
pheno = np.array([[0 for x in range(2)] for y in range(GCKD_con.shape[0])],dtype='int') # empty array
pheno[:,0]=np.copy(GCKD.iloc[:,0]) # copy pheno data and swap data in next column
for i in range(pheno.shape[0]) :
   if (pheno[i,0] == 0) :
     pheno[i,1] = 1
   else :
     pheno[i,1] = 0
```



```
>>> pheno[0:3,:]
array([[1, 0],
       [0, 1],
       [1, 0]])
>>>
```

# When we have in addition 5 batches as additional discrete variables 5 additional columns required.
# See also example test set 1.
# Also, compare to Limma design matrix

# Levels for discrete data
# We have only sex with male = 1 and female = 0, therefore, levels = 2
GCKD_levels=np.array([2])

```
# When we have in addition 5 batches as additional discrete variables array should look like this: [2,5]
# See also example test set 1.
```

# Lambda

```
# For the actual calculations using lasso regression we have to define a sequence of possible
# lambdas. Here we follow the description given in Altenbuchinger et al., Sci. Rep. 2019, 9:13954 |
# https://doi.org/10.1038/s41598-019-50346-2, supplement 1.4 'Calibration of the penalty
# parameter lambda'. Note with the following code we do # not set a specific lambda we only
provide a set of possible values which will be tried in the next
# steps to select the optimal value amongst them. Lam-zero depends on both the number of
# continuous and discrete features and the number of samples.
# as described below you may have to adept this sequence to your data. Note, in Altenbuchinger et
# al. 2019 the following sequence (lam_seq) was swept 2; 1:75; 1:5; 1:25; . . . ; -4:5; -4:75; -5.
# However, more values in the sequence may cause considerably longer computation times in the
# leave one out cross validation, where each lambda value is tried separately. Note, that for data
# imputation and following cross validation different lambda values may be optimal which may
# require an adaption of lam_seq to find in each case the optimal lambda.

lam_zero = np.sqrt(np.log(GCKD_con_s.shape[1] + pheno.shape[1]/2)/GCKD_con_s.shape[0])
lam_seq = np.array([-1.75,-2.0,-2.25]) # may need adaption
lam = [pow(2, x) for x in lam_seq]
lam = np.array(lam)
lam = lam_zero * lam
# With 'print(lam)' you display the sequence of possible lambda values
```

# Impute missing values with ADMIRE

```
# Both continuous and discrete variables will be imputed, in this example we have only missing
# continuous data
GCKD_con_s_imp, pheno_imp,lam_o = impute(GCKD_con_s,pheno,GCKD_levels,lam)
print(np.sum(np.isnan(GCKD_con_s_imp))) # 0 to check if all continuous missing values have been
# successfully imputed
# 'print(lam_o)' displays the optimal lamda, it should be in the middle of your possible lambda values
```

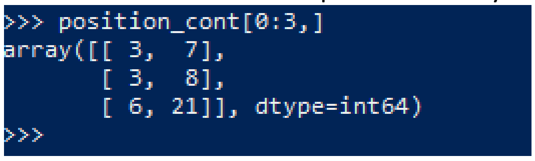
# Perform cross validation in ADMIRE to determine optimal lambda and build MGMs

```
prob_hat, B_m, lam_opt,  GCKD_con_s_imp_hat, pheno_imp_hat =
loo_cv_cor(GCKD_con_s_imp,pheno_imp,GCKD_levels,lam)
# in this routine the lambda value is independently optimized.
# 'print(lam_opt)' shows this value again it should be in the middle of the possible lambda values
# If this is not the case change and/or increase the sequence of possible lambda values so that both
# optimal values are not at the border of the sequences. For example if the optimal lambda
# corresponds to the first value of your sequence the optimal value has not been reached.
```

# Determine continuous threshold and perform correction in ADMIRE accordingly

```
# Return: continuous data corrected for detected anomalies, threshold, number of detected
anomalies (n_ano_cont) and position
GCKD_con_s_imp_cor, threshold_cont, n_ano_cont, position_cont =
get_threshold_continuous(GCKD_con_s_imp, GCKD_con_s_imp_hat, B_m)
print(n_ano_cont) # 726 detected continuous anomalies
print(threshold_cont) # threshold for detected anomalies e.g. 1.95
print(position_cont)
```

# gives positions of all detected continuous anomalies. eg. '6, 0'. Note, row, column, numbering # starts with zero. This is important when you display your corrected data in e.g. Excel



```
>>> position_cont[0:3,]
array([[ 3,  7],
       [ 3,  8],
       [ 6, 21]], dtype=int64)
>>>
```

# e.g. 'GCKD_con_s_imp[6,0]'=6.97; 'GCKD_con_s_imp_cor[6,0]'=1.69; in case of no anomaly no correction e.g. 'GCKD_con_s_imp[0,0]'=0.119, GCKD_con_s_imp_cor[0,0]'=0.119

# Format back conversion

```
# Note at this point corrected continuous data are in standardized format often you would like to
# revert back to original non-standardized format for example to manually check for measurement
# errors
GCKD_con_imp_cor= np.array([[0 for x in range(GCKD_con_s_imp_cor.shape[1])] for y in range(GCKD_con_s_imp_cor.shape[0])],dtype='float') # create array for back transformed data
# remember each metabolite is one column, loop through all metabolites
for i in range(GCKD_con.shape[1]) :
    mean = np.nanmean(GCKD_con[:,i])
    var = np.nanvar(GCKD_con[:,i])
    std = var ** 0.5
    real_val = GCKD_con_s_imp_cor[:,i] * std + mean
    GCKD_con_imp_cor[:,i] = real_val
```

# Negative predictions

```
# In some cases, negative predictions occur (mostly values very close to zero) this clearly must be
# wrong set all negative values to 'nan'
GCKD_con_imp_cor[GCKD_con_imp_cor < 0] = float("nan")
```

# Add row- and col-names

```
GCKD_con_imp_cor =
pd.DataFrame(GCKD_con_imp_cor,list(GCKD.index),columns=list(GCKD.columns[1:GCKD.shape[1]]))
# 'GCKD_con_imp_cor' now contains the non-standardized corrected values. Note all values with
# more than 4 decimals were either corrected or imputed by ADMIRE
```

# Write to .csv

```
GCKD_con_imp_cor.to_csv('C:/Users/Username/Metabolomics/Statistics/Outlier_MGM/Test_data/GCKD/GCKD_con_imp_cor.csv',decimal=',',sep=';')
```

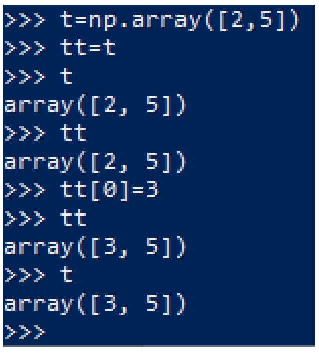
# Discrete anomalies

```
# Discrete anomalies would correspond here to wrongly labeled sex identifiers
n_ano_disc, threshold_cont, position_disc = get_threshold_discrete(pheno, GCKD_levels, pheno_imp_hat)
# returns:  number of detected anomalies (n_ano_disc), threshold and position
print(n_ano_disc)
# 0 detected discrete anomalies in this example. In case of discrete anomalies their positions will be
# stored in 'position_disc'
```

****

# Note of caution when working with Python

```
# Copying of data arrays
# The command ‘tt=t’ does not copy the data of the array ‘t’ but it
# copies only the address of the array ‘t’. So now we have two
# addresses that point to the same data. The data itself are only stored
# once on your computer. To copy the data use for example ‘np.copy’
```



```
>>> t=np.array([2,5])
>>> tt=t
>>> t
array([2, 5])
>>> tt
array([2, 5])
>>> tt[0]=3
>>> tt
array([3, 5])
>>> t
array([3, 5])
>>>
```