DSpectra_scripts

February 8, 2018

In [1]:
```
# Import modues
# -*- encoding: utf-8 -*-
import numpy as np
import matplotlib as mpl
from matplotlib import rc
import math
import pandas as pd
import os
import itertools
from scipy import stats
from scipy import ndimage
import seaborn as sns

import matplotlib as mpl
from matplotlib import cm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

from statsmodels.stats.descriptivestats import sign_test
from statsmodels.stats.weightstats import zconfint
from statsmodels.stats.weightstats import *

from skimage import measure
from scipy import ndimage
from scipy import misc

from scipy.stats.stats import pearsonr, spearmanr
from collections import  Counter

# from pandas import ExcelWriter
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import statsmodels.stats.api as sm
```

```
from sklearn import cross_validation, datasets, linear_model, metrics
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import OneHotEncoder

from sklearn.cluster import KMeans
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn import cross_validation, datasets, grid_search, linear_model, metrics
from sklearn.metrics import classification_report

from scipy.optimize import curve_fit
from scipy import signal

from sklearn import random_projection
from sklearn.decomposition import RandomizedPCA
from sklearn.decomposition import PCA
from sklearn import manifold
from sklearn.cluster import KMeans

sns.set_style("whitegrid")
# flatui = ["#9b59b6", "#3498db", "#95a5a6", "#e74c3c", "#34495e", "#2ecc71"]
sns.set_palette('Accent')

rc('font', family='Arial', size=14)  # change font for russian

rc('xtick', labelsize=14)
rc('ytick', labelsize=14)
mpl.rcParams.update({'font.size': 14})

% matplotlib inline
print 'Import Ready'
```

```
Import Ready


/usr/local/lib/python2.7/dist-packages/sklearn/cross_validation.py:41: DeprecationWarning: This
  "This module will be removed in 0.20.", DeprecationWarning)
/usr/local/lib/python2.7/dist-packages/statsmodels/compat/pandas.py:56: FutureWarning: The panda
  from pandas.core import datetools
/usr/local/lib/python2.7/dist-packages/sklearn/grid_search.py:42: DeprecationWarning: This modul
  DeprecationWarning)
```

1 Spectrum parameters

In [2]:
```
markers = ['o','s','d','v','h','>','<','*','^','+']
markers_line = ['-o','-s','-d','-v', '-h', '->','-<', '-*']
```

```
        cmap = plt.get_cmap('Accent')
        colors = [cmap(i) for i in np.linspace(0., 1., 8)] * 2

In [3]: # Function for spectra features
        def smooth (spec_i, params, type_smooth):
            # type 1 - average filter, type 2 - SavGol, type 3 - FFT
            '''
            Function for smoothing spectra. Return smooth spectra as np.array of intencity (np.a
            Parameters:
                spectra (list of intence)
                parameters of smooth (optimizing):
                    type 1: medians - [window (full size, int)]
                    type 2: average - [window (full size, int)]
                    type 3: SavGol - [polinom order (int), windows (full size, odd, int)]
                    type 4: Discret Furie - [min frequence (float [0, 2])]
                type smooth (see above)
            '''
            spec_i = np.array(spec_i)
            spec_tranc = []
            if type_smooth == 0:
                # median filter w - full window size
                window = params[0]
                spec_i = np.array(spec_i)
                for i in xrange(len(spec_i)):
                    d = None
                    if i < window:
                        d = np.median(spec_i[0:i+window])
                    elif i + window > len(spec_i):
                        d = np.median(spec_i[i-window:])
                    else:
                        d = np.median(spec_i[int(i-window/2):int(i+window/2)])
                    spec_tranc.append(d)
            elif type_smooth == 1:
                # average filter w - full window size
                window = params[0]
                spec_i = np.array(spec_i)
                for i in xrange(len(spec_i)):
                    d = None
                    if i < window:
                        d = np.average(spec_i[0:i+window])
                    elif i + window > len(spec_i):
                        d = np.average(spec_i[i-window:])
                    else:
                        d = np.average(spec_i[int(i-window/2):int(i+window/2)])
                    spec_tranc.append(d)
            elif type_smooth == 2:
                # Sav-Gol filter
```

```
            deriv = 0
            rate = 1
            order = params[0]
            window_size = params[1]  # full size
            if window_size % 2 != 1 or window_size < 1:
                print 'window_size size must be a positive odd number'
                return
            if window_size < order + 2:
                print 'window_size is too small for the polynomials order'
                return
            order_range = xrange(order+1)
            half_window = (window_size - 1) // 2
            b = np.mat([[k**i for i in order_range] for k in xrange(- half_window, half_wind
            m = np.linalg.pinv(b).A[deriv] * rate**deriv * math.factorial(deriv)

            firstvals = spec_i[0] - np.abs(spec_i[1:half_window+1][::-1] - spec_i[0])

            lastvals = spec_i[-1] + np.abs(spec_i[-half_window-1:-1][::-1] - spec_i[-1])
            spec_i = np.concatenate((firstvals, spec_i, lastvals))
            spec_tranc = np.convolve(m[::-1], spec_i, mode='valid')

            lendelta = len(spec_tranc) - len(spec_i)
            if lendelta > 0:
                print 'lendelta: ', lendelta
                spec_tranc = spec_tranc[lendelta // 2:]
        else:
            # Calculate furie-conversion os spec
            fft_res = np.fft.fft(spec_i)
            u_crit = params[0]
            n = float(len(fft_res))  # number of channels
            filtered = map(lambda x: 0 if (x / n > u_crit) else fft_res[x], xrange(len(fft_r
            spec_tranc = map(lambda x: x.real, np.fft.ifft(filtered))
        return np.array(spec_tranc)


    # Autofind peacks and base line
    def calc_h(i, w, v):
        '''
        Calulate h parameters for 'zero-filter' algorithm. rturn h value (float).
        Parameters:
            i - channel value [0, 4095] (int)
            w - windows (even, int)
            v - windows shift (int)
        '''
        h = None
        if (-v - w / 2) <= i < -w / 2:
            h = -0.5 / v
        elif -0.5*w <= i <= 0.5*w:
```

```
        h = 1. / w
    elif 0.5*w < i <= (0.5*w+v):
        h = - 0.5 / v
    else:
        print 'Not good: ', i
    if h == 0: print 'WTF'
    return h


def zero_area(spec, params, type_alg):
    # type_alg = 1 - classical zero-area algorithm, 2, 3 - my algo as float average and
    '''
    Function for 'zero-area'. Return mask of base line as np.array of baseline intence (
    Parameters:
        spectra (list of intence)
        parameters (optimizing):
            type 1: classical - [window (even, int), window_shift (int)]
            type 2: my as average - [window (int)] - work bad
            type 3: my as parabola - [windows (int)] - work bad
        type smooth (see above)
    '''
    spec_transform = []
    spec = np.array(spec)
    if type_alg == 1:
        w = params[0]
        v = params[1]
        if w % 2 != 0:
            print 'w mast be even'  #
            return
        for i in xrange(len(spec)):
            i = int(i)
            d = 0
            p1 = -v-0.5*w
            p2 = +v+0.5*w
            j = int(p1)
            if i + p1 < 0:
                while j != p2:
                    if i + j < 0:
                        d += calc_h(j, w, v) * spec[0]
                    else:
                        d += calc_h(j, w, v) * spec[int(i+j)]
                    j+=1
            elif i + p2 > len(spec):
                while j != p2:
                    if i + j >= len(spec):
                        d += calc_h(j, w, v) * spec[-1]
                    else:
                        d += calc_h(j, w, v) * spec[int(i+j)]
```

```
                j+=1
        else:
            while j != p2:
                d += calc_h(j, w, v) * spec[int(i+j)]
                j+=1
        spec_transform.append(d)
elif type_alg == 2:
    # Can write the same algo for smoothind filters type 0-2
    chanel = xrange(len(spec))
    w = params[0]  # windows full-lenght
    spec_transform = range(len(spec))
    num_w = int(len(spec) / w)  # numbers of subspec for find peaks
    is_peak_up = [0] * num_w
    is_peak_out = [0] * num_w
    spec_parts = []
    # write cond for peak up and down
    for i in xrange(num_w):
        start_i = w * i
        stop_i = w * (i + 1)
        if stop_i > len(spec): stop_i = len(spec)
        spec_i = np.array(spec[start_i: stop_i])
        spec_parts.append(spec_i)
        aver_i = np.average(spec_i)  # may be use median? - , ..
        cond_i = spec_i > aver_i
        if np.sum(cond_i) > w / 2:  # cond for peak
            if np.sum(cond_i[:w/2]) > np.sum(cond_i[w/2:]):
                is_peak_out[i] = 1
            else:
                is_peak_up[i] = 1
    for i in xrange(num_w-3):
        d = is_peak_up[i]
        if d and np.sum(is_peak_out[i:i+3]):
            for j in xrange(len(spec_parts[i])):
                spec_transform[i+j] = 0
        else:
            for j in xrange(len(spec_parts[i])):
                spec_transform[i+j] = spec_parts[i][j]
    spec_transform.append([0 for x in xrange((num_w-3)*w)])
elif type_alg == 3:
    deriv = 0
    rate = 1
    order = 4
    window_size = params[0]  # full size
    if window_size < order + 2:
        print 'window_size is too small for the polynomials order'
        return

    chanel = xrange(len(spec))
```

```
            spec_transform = range(len(spec))

            num_w = int(len(spec) / window_size)  # numbers of subspec for find peaks
            spec_parts = [spec[x*window_size:(x+1)*window_size] for x in xrange(num_w-1)]
            for i in xrange(len(spec_parts)):
                # aprox point as 2nd order polinom and calculate a and R2
                raw_spec = spec_parts[i]
                coef, covar, R2 = approx_poly(xrange(len(raw_spec)), raw_spec, 2)
                a = coef[0]
                if a < 0 and R2 > 0.7:
                    # is peak
                    for j in xrange(len(spec_parts[i])):
                        spec_transform[i+j] = 0
                else:
                    for j in xrange(len(spec_parts[i])):
                        spec_transform[i+j] = spec_parts[i][j]
        return np.array(spec_transform[:4095])


    def is_baseline(spec_tr, floor = -1):
        # return True/false mask for baseline
        '''
        Function for baseline mask. Return mask as np.array of boolen (np.array)
        Parameters:
            spectra of baseline (list of intence)
            parameters (optimizing):
                medians - [flag of maximum base intence (int or float)]
        '''
        spec_tr = np.abs(spec_tr)
        flag = floor
        if floor < 0:
            flag = np.average(spec_tr)
        spec_res = spec_tr < flag
        for i in xrange(10,len(spec_res)-10):
            if spec_res[i] and np.sum(spec_res[i-10:i+10]) < 10:
                spec_res[i] = False
        return spec_res


    # Calculate finish parameters
    def approx_poly(list_x, list_y, order):
        # calculate coef for polinom approximation [A, B], y = Ax + B return highest power f
        '''
        Function for approximate points as order poinom. Return approximate coeffisients? co
        Parameters:
            list of x points
            list of y points
            order of polinom
```

```
    '''
    poly_coeff = np.polyfit(list_x, list_y, order, cov=True)
    R2 = None
    # r-squared for 1 order  TODO: add Matius coefficient
    if (order - 1.0) < 1.5:  # use 0.5 for 1st order polinom
        p = np.poly1d(poly_coeff[0])
        # fit values, and mean
        yhat = p(list_x)                          # or [p(z) for z in x]
        ybar = np.sum(list_y)/len(list_y)           # or sum(y)/len(y)
        ssreg = np.sum((yhat-ybar)**2)   # or sum([ (yihat - ybar)**2 for yihat in yhat]
        sstot = np.sum((list_y - ybar)**2)    # or sum([ (yi - ybar)**2 for yi in y])
        R2 = ssreg / sstot
        R2 = round(float(R2), 5)
    return poly_coeff[0], poly_coeff[1], R2


def cur_poly_value(x, coef):
    # return y for x and coef of polinom, coef highest power first
    '''
    Function for calculate point with polinoms coefficients. Return value of polinom (fl
    Parameters:
        point (int)
        coefficient (highest pover first, list)
    '''
    res = 0.0
    cur_x_dg = 1.0
    for i in xrange(1, len(coef)+1):
        res += cur_x_dg * coef[-i]
        cur_x_dg *= x
    return res


def calc_all_area(spec_i):
    # return area of spectra
    '''
    Function for calculate area of baseline. Return data (float).
    Parameters:
        baseline spectra of intencity (list)
    '''
    return np.sum(spec_i)


def calc_baseline(spec_i, baseline_mask):
    # return Imax ans S of background
    '''
    Function for approximate baseline as 5th order poinom. Return approximate baseline a
    Parameters:
        spectra (list of intence)
```

```
        baseline mask (list of boolen)
    '''
    spec_i = np.array(spec_i)
    x = np.array([x for x in xrange(len(spec_i))])
    poly_coef = approx_poly(x[baseline_mask], spec_i[baseline_mask], 5)[0]
    approx_baseline = []
    for i in x:
        approx_baseline.append(cur_poly_value(i, poly_coef))
    approx_baseline = np.array(approx_baseline)
    return approx_baseline


def calc_peacs(spec_i, base_fit, baseline_mask, parameters = None, type_calc=2, energy =
    # return i_max, S and channel of peack
    '''
    Function for calculate peak data.
    Return:
        intencity of peaks (list)
        cannels of peaks (list)
        gauss intencity of each peaks (list)
        gayss squares of each peaks (list)
        intencity of fitting base line (list)
    Parameters:
        spectra (list of intence)
        baseline (list of intence)
        baseline mask (list of boolen)
        parameters (for optimisation):
            type 1 (derivative): [iteration for smooth of diff spectra [0, 10], treshold
            type 2 (zero-area base): [iteration of smooth (int)]
        type (see above)
        energy of close peak (treshold parameters, float 0.2)
        a0, a1 - channel to energy parameters (float)
    '''
    spec_i = np.array(spec_i)
    max_delta_chan = None
    if a0 is not None and a1 is not None:
        max_delta_chan = (energy - a0) / a1  # for close peaks
    else:
        max_delta_chan = (energy - spe_df['a0'][0]) / spe_df['a1'][0]  # for close peaks

    if type_calc == 1:
        # derivative type - for all spectra
        itera = parameters[0]  #
        der_threshold = parameters[1]  # 10 for work - oprimize
        spec_new = np.diff(spec_i, 2)
        while itera > 0:
            spec_new = smooth(spec_new, params=[3, 11], type_smooth=2)  # Sav-Gol
            itera -= 1
```

```
        der = np.array(spec_new)
        peak_mask = der < der_threshold  # peak search
        i_all = []
        channel_all = []
        delta_peak = 0
        for i in xrange(len(peak_mask)-1):
            # search peaks channel
            if peak_mask[i] and peak_mask[i+1]:  # TODO: set threshold longer then i+1 ?
                # if peak prolong
                delta_peak += 1
            elif peak_mask[i]:
                # end peak
                if delta_peak > 0:
                    i_max = np.max(spec_i[i-delta_peak:i])  # max intence
                    c = np.where(spec_i[i-delta_peak:i] == i_max)[0][0]  # channel of ma
                    i_max = np.average(spec_i[i-delta_peak+c-3:i-delta_peak+c+3])  # cal
                    i_all.append(i_max)
                else:
                    pass
            else:
                # no peak
                delta = 0
        # calculate square of peak and (I - base line)
        i_all = np.array(i_all)
        gauss_i, gauss_s, base_i = gauss_peaks(i_all, channel_all, base_fit, spec_i)
    elif type_calc == 2:
        w0 = 3  # parameters[0]  # SG polinom order
        w1 = 11  # parameters[1]  # SG window size
        itera = parameters[0]  # numbers of iteration - optimize, 2
        x = np.array(xrange(len(spec_i)))
        x_peak = x[~baseline_mask]  # mask for peaks

        spec_new = []
        for i in xrange(len(spec_i)):
            # for new 'spec' with peak only
            if i in x_peak:
                spec_new.append(spec_i[i])
            else:
                spec_new.append(0)
        while itera > 0:
            spec_new = smooth(spec_new, params=[w0, w1], type_smooth=2)  # Sav-Gol
            itera -= 1
        channel_all = (np.diff(np.sign(np.diff(spec_new))) < 0).nonzero()[0] # + 1  # lo

        spec_new = np.array(spec_new)
        i_all = spec_new[channel_all]
        i_all = np.array(i_all)
        channel_all = channel_all[i_all > 1.]
```

```
        i_all = i_all[i_all > 1.]
        gauss_i, gauss_s, base_i =  gauss_peaks(i_all, channel_all, base_fit, spec_i)
    return i_all, channel_all, gauss_i, gauss_s, base_i


def gaus(x,a,x0,sigma):
    # for gaussian fit
    '''
    Function for gaussian curve. Return gauss data (int or np.array)
    Parameters:
        x point
        aplitude
        senter of peak
        std of peak
    '''
    return a * np.exp(-(x - x0)*(x - x0) / (2 * sigma*sigma))


def gauss_peaks(i_all, channel_all, base_fit, spec_i):
    # calculate Gauss S and I of peaks
    '''
    Function for gaussian approximation of peaks.
    Return:
        intens of peaks (np.array)
        squares of peaks (np.array)
        intence of base line (np.array)
    Parameters:
        intence of peaks (list)
        channels of peaks (list)
        intence of fit baseline (list)
        intence of raw spectra (list)
    '''
    base_i = []
    gauss_s = []
    gauss_i = []
    for i in xrange(len(channel_all)):
        c_middle = channel_all[i]
        base_i.append(np.average(base_fit[c_middle]))
        g_h = 0
        g_s = 0
        w = 15
        y = np.array(spec_i[c_middle-w:c_middle+w])  # gauss fit of points
        x = np.array(xrange(len(y)))
        if y.shape is not () and x.shape is not ():
        # correction for weighted arithmetic mean
            mean = np.sum(x * y) / np.sum(y)
            sigma = np.sqrt(np.sum(y * (x - mean)**2) / np.sum(y))
            try:
```

```
                    popt,pcov = curve_fit(gaus, x, y, p0=[max(y), mean, sigma])
                    y_fit = gaus(x, *popt)
                    g_h = np.max(y_fit)
                    g_s = g_h * (2 * np.pi) ** 0.5 * popt[-1]  # gauss square
                    w += 2
                except RuntimeError as var:
                    g_h = 0
                    g_s = 0
                except ValueError as var:
                    print var
                    print x, y
                    g_h = 0
                    g_s = 0
            else:
                g_h = 0
                g_s = 0
            gauss_i.append(g_h)
            gauss_s.append(g_s)


        return np.array(gauss_i), np.array(gauss_s), np.array(base_i)
```

In [4]:
```
# load spec data
def dir_name(_):
    _ = _[0].split('\\')
    if len(_) > 1: return 'specs/' + _[1]

dirs = ['specs/' + x for x in list(os.walk('specs'))[0][1]]

dict_obj_spe = {}

directs = dirs
shifr = dict(zip(range(len(directs)), directs))
counter = 0
for dir_name in directs:
    if dir_name:
        print 'Files in directory: ', dir_name, len(os.listdir(dir_name))
        data_num = xrange(len(os.listdir(dir_name)))
        n = dir_name.split('/')[1]
        n = n.split('_')

        ftype = n[0].lower()
        mark = n[1]
        fraction =n[2]
        kd = n[3]  # binary yes/no

        if len(kd.split('.')) > 1:
            kd = kd.split('.')[1]  # float or none - undefine
```

```
for i in xrange(len(os.listdir(dir_name))):
    dict_obj_spe[counter] = {}
    i = os.listdir(dir_name)[i]
    name =  str(i)
    if '.spe' in i:
        spec_file = open(dir_name + '/' + i)

        spec_num = i.split('_')[-1][0]  # select one number
        if spec_num in ['1', '2']:
            _ = i.split('_')[-1][1]
            if _ not in ['(', '.']:
                spec_num += _
        if ftype == 'npss.zn':
            spec_num = i.split('_')[-2] + '_' + spec_num
        if ftype == 'an':
            spec_num = '_'.join(i.split('_')[1:4])
            spec_num = spec_num[:-4]
        if spec_num in ['(','.','00','k','n']:
            print i, ' ', spec_num, ' ', ftype
        lines = spec_file.readlines()
        spec_file.close()

        # properties
        exposition = None
        voltage = None
        anod_current = None
        atmo = None
        list_e = []
        list_i = []

        try:
            exposition = int(lines[3])
            voltage = int(lines[6])
            anod_current = int(lines[7])
            atmo = int(lines[17])
        except ValueError as var:
            print ('ERROR: ', var)
        a0 = float(lines[19])
        a1 = float(lines[20])
        a2 = float(lines[21])

        lines_spec = tuple(lines[32:])

        c = 0
        for ind in lines_spec:
            e = a0 + a1*c + a2*c*c
            list_e.append(e)
```

```
                            list_i.append(int(ind))
                            c += 1

                        dict_obj_spe[counter]['exposition'] = exposition
                        dict_obj_spe[counter]['voltage'] = voltage
                        dict_obj_spe[counter]['current'] = anod_current
                        dict_obj_spe[counter]['atmo'] = atmo
                        dict_obj_spe[counter]['ftype'] = ftype
                        dict_obj_spe[counter]['mark'] = mark
                        dict_obj_spe[counter]['spec_num'] = spec_num
                        dict_obj_spe[counter]['fraction'] = fraction
                        dict_obj_spe[counter]['kd'] = kd
                        dict_obj_spe[counter]['a0'] = a0
                        dict_obj_spe[counter]['a1'] = a1
                        dict_obj_spe[counter]['a2'] = a2
                        dict_obj_spe[counter]['energy'] = list_e
                        dict_obj_spe[counter]['intence'] = list_i
                        counter += 1
        print ('Done')
```

```
Files in directory:  specs/NPKS_4.30.15.16_rawgrain_kd.None 10
Files in directory:  specs/NPKS_4.30.15.16_grain_kd.None 24
Files in directory:  specs/NPKS_4.30.15.16_100_kd.None 29
Files in directory:  specs/NPKS_4.30.15.16_500_kd.None 26
Files in directory:  specs/NPKS_4.30.15.16_500.dry_kd.None 15
Files in directory:  specs/NPKS_4.30.15.16_100.dry_kd.None 14
Done
```

```
In [5]: # Create df
        spe_df = pd.DataFrame.from_dict(dict_obj_spe, orient='index')
```

1.1 -

```
In [6]: # smoothing spectra

        test_df = spe_df[spe_df.fraction == 'grain']
        ftype_all = Counter(test_df.mark)
        print ftype_all
        aver_spec_type = {}

        smooth_type = {
            0:xrange(2,23,5),
            1:xrange(2,23,5),
            2:[xrange(2,6), (7,9,15,21)],
            3:np.arange(1, 150, 20) / 100.
        }
```

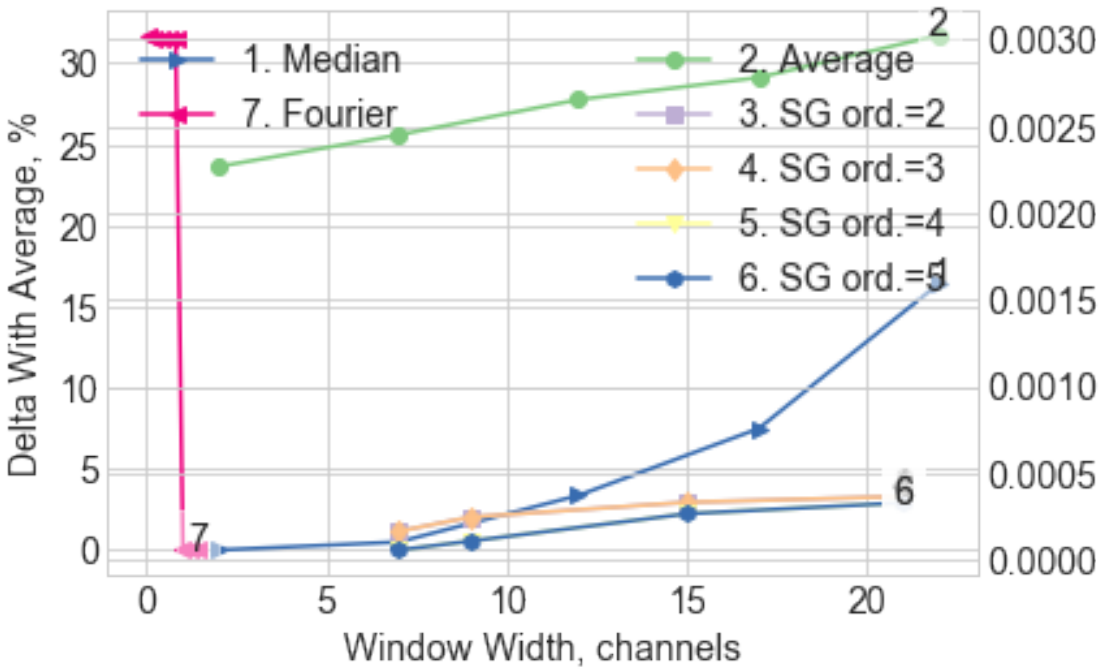
```
        data_smooth1 = {}

        for ftype in ftype_all.keys():
            # minimaise delta with average spectra for each type
            test_df2 = test_df[test_df.mark == ftype]
            spec_all = test_df2.intence
            aver_spec = np.mean(np.array([np.array(x) for x in spec_all]), axis=0)
            aver_spec_type[ftype] = aver_spec
            data_smooth1[ftype] = {}

            # smooth
            for t in smooth_type.keys():
                print ftype, 'smooth type: ', t,
                data_smooth1[ftype][t] = {}
                if t != 2:
                    data_smooth1[ftype][t]['no'] = {}
                    for p in smooth_type[t]:
                        s = []
                        for spec_i in test_df2.intence:
                            s.append(np.array(smooth(spec_i, [p], type_smooth=t)))
                        now_mean = np.mean(np.array([np.array(x) for x in s]), axis=0)
                        delta = (aver_spec - now_mean) / np.average(aver_spec) * 100.
                        data_smooth1[ftype][t]['no'][p] = np.mean(delta[:2500])
                else:
                    data_smooth1[ftype][t] = {}
                    for p in smooth_type[t][0]:
                        data_smooth1[ftype][t][p] = {}
                        for p2 in smooth_type[t][1]:
                            s = []
                            for spec_i in test_df2.intence:
                                s.append(np.array(smooth(spec_i, [p, p2], type_smooth=t)))
                            now_mean = np.mean(np.array([np.array(x) for x in s]), axis=0)
                            delta = (aver_spec - now_mean)/ np.average(aver_spec) * 100.
                            data_smooth1[ftype][t][p][p2] = np.mean(delta[:2500])
                print 'done'
        print 'All Done'
```

```
Counter({'4.30.15.16': 24})
4.30.15.16 smooth type:  0 done
4.30.15.16 smooth type:  1 done
4.30.15.16 smooth type:  2 done
4.30.15.16 smooth type:  3 done
All Done
```

```
In [7]: # aver_spec_type - include average spec for first key
        mpl.rcParams.update({'font.size': 14})
        for f in data_smooth1:
```

```
# Create structure for plot
counter = 1
metrik_data=[]
x_data = []
metrik_name = []
for a in data_smooth1[f]:
    if a in [0,1,3]:
        temp_d = []
        temp_x = []
        for p2 in data_smooth1[f][a]['no']:
            temp_d.append(data_smooth1[f][a]['no'][p2])
            temp_x.append(p2)
        temp_x, temp_d = (list(x) for x in zip(*sorted(zip(temp_x, temp_d), key=lamb
        metrik_data.append(temp_d)
        x_data.append(temp_x)
        if a == 1:
            metrik_name.append(str(counter) + u'. Average')
        elif a == 0:
            metrik_name.append(str(counter) + u'. Median')
        else:
            metrik_name.append(str(counter) + u'. Fourier')
        counter += 1
    else:
        for p1 in data_smooth1[f][a]:
            temp_d = []
            temp_x = []
            for p2 in data_smooth1[f][a][p1]:
                temp_d.append(data_smooth1[f][a][p1][p2])
                temp_x.append(p2)
            temp_x, temp_d = (list(x) for x in zip(*sorted(zip(temp_x, temp_d), key=
            metrik_data.append(temp_d)
            x_data.append(temp_x)
            metrik_name.append(str(counter) + u'. SG ord.='+str(p1))
            counter += 1

# create structure for two axis
metrik_data1, metrik_data2 = [], []
x_data1, x_data2 = [], []
metrik_name1, metrik_name2 = [], []
for i in xrange(len(metrik_data)):
    if u'SG' in metrik_name[i] or u'. Average' in metrik_name[i]:
        metrik_data1.append(metrik_data[i])
        x_data1.append(x_data[i])
        metrik_name1.append(metrik_name[i])
    else:
        metrik_data2.append(metrik_data[i])
        x_data2.append(x_data[i])
        metrik_name2.append(metrik_name[i])
```

```
c=0

fig, ax1 = plt.subplots()
ax2 = ax1.twinx()
for i in xrange(len(metrik_data1)):
    y = metrik_data1[i]
    x = x_data1[i]
    ax2.plot(x, y, markers_line[c], label=metrik_name1[i], c=colors[i])
    ax2.text(x[-1], y[-1], metrik_name1[i].split('.')[0],
          horizontalalignment='center',
          bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))
    c+=1

for j in xrange(len(metrik_data2)):
    y = metrik_data2[j]
    x = x_data2[j]
    ax1.plot(x, y, markers_line[c], label=metrik_name2[j], c=colors[j+len(metrik_nam
    ax1.text(x[-1], y[-1], metrik_name2[j].split('.')[0],
          horizontalalignment='center',
          bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))
    c+=1

ax1.set_xlabel(u'Window Width, channels')
ax1.set_ylabel(u'Delta With Average, %')
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')
plt.draw()
plt.savefig(f+' spec_smooth_aver_eng.png', dpi=300)
plt.show()
```



In [8]:
```
# std-optimization

test_df = spe_df[spe_df.fraction == 'grain']
ftype_all = Counter(test_df.ftype)
print ftype_all
aver_spec_type = {}

smooth_type = {
    0:xrange(2,23,5),
    1:xrange(2,23,5),
    2:[xrange(2,6), (7,9,15,21)],
    3:np.arange(1, 150, 20) / 100.
}

data_smooth2 = {}

for ftype in ftype_all.keys():
    # minimaise delta with average spectra for each type
    test_df2 = test_df[test_df.ftype == ftype]
    spec_all = test_df2.intence
    aver_std = np.std(np.array([np.array(x) for x in spec_all]), axis=0)
    aver_spec_type[ftype] = aver_std
    data_smooth2[ftype] = {}
    print np.average(aver_std)
```

```
            # smooth
            for t in smooth_type.keys():
                print ftype, 'smooth type: ', t,
                data_smooth2[ftype][t] = {}
                if t != 2:
                    data_smooth2[ftype][t]['no'] = {}
                    for p in smooth_type[t]:
                        s = []
                        for spec_i in test_df2.intence:
                            s.append(np.array(smooth(spec_i, [p], type_smooth=t)))
                        std_spec_i = np.std(s, axis=0)
                        data_smooth2[ftype][t]['no'][p] = np.average(std_spec_i, axis=0) / np.av
                else:
                    data_smooth2[ftype][t] = {}
                    for p in smooth_type[t][0]:
                        data_smooth2[ftype][t][p] = {}
                        for p2 in smooth_type[t][1]:
                            s = []
                            for spec_i in test_df2.intence:
                                s.append(np.array(smooth(spec_i, [p,p2], type_smooth=t)))
                            data_smooth2[ftype][t][p][p2] = np.average(np.std(s, axis=0),axis=0)
                print 'done'
        print 'All Done'
```

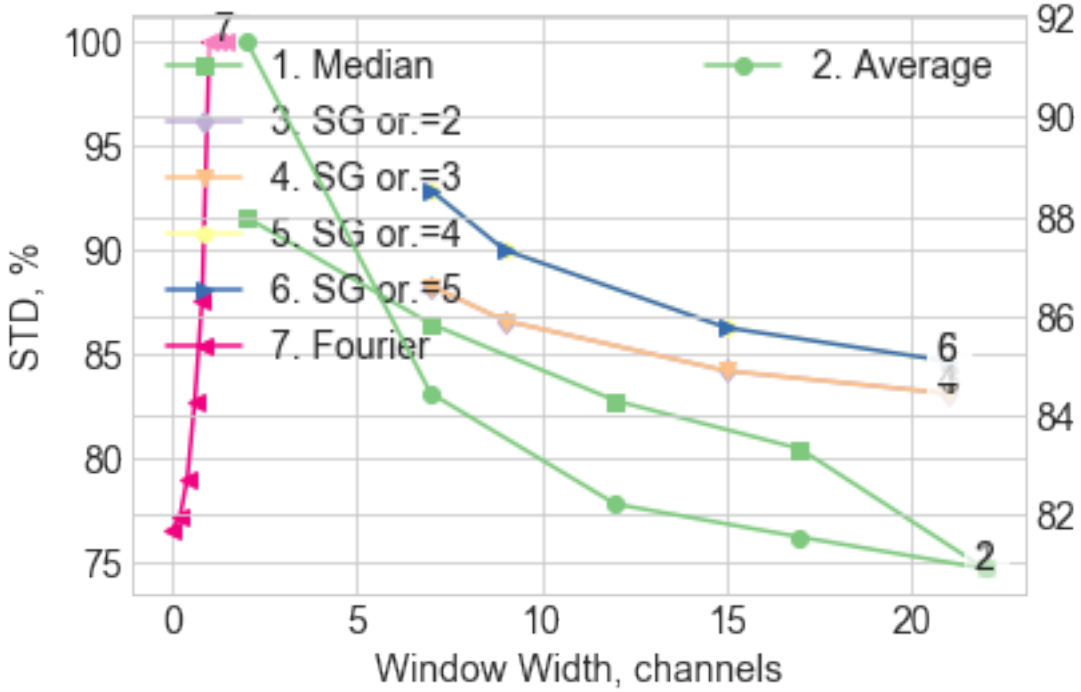
```
Counter({'npks': 24})
19.621613426299703
npks smooth type:  0 done
npks smooth type:  1 done
npks smooth type:  2 done
npks smooth type:  3 done
All Done
```

In [9]:
```
mpl.rcParams.update({'font.size': 14})
```

```
        for f in data_smooth2:
            # Create structure for plot
            counter = 1
            metrik_data=[]
            x_data = []
            metrik_name = []
            for a in data_smooth2[f]:
                if a in [0,1,3]:
                    temp_d = []
                    temp_x = []
                    for p2 in data_smooth2[f][a]['no']:
                        temp_d.append(data_smooth2[f][a]['no'][p2])
                        temp_x.append(p2)
```

```
            temp_x, temp_d = (list(x) for x in zip(*sorted(zip(temp_x, temp_d), key=lamb
            metrik_data.append(temp_d)
            x_data.append(temp_x)
            if a == 1:
                metrik_name.append(str(counter) + u'. Average')
            elif a == 0:
                metrik_name.append(str(counter) + u'. Median')
            else:
                metrik_name.append(str(counter) + u'. Fourier')
            counter += 1
        else:
            for p1 in data_smooth2[f][a]:
                temp_d = []
                temp_x = []
                for p2 in data_smooth2[f][a][p1]:
                    temp_d.append(data_smooth2[f][a][p1][p2])
                    temp_x.append(p2)
                temp_x, temp_d = (list(x) for x in zip(*sorted(zip(temp_x, temp_d), key=
                metrik_data.append(temp_d)
                x_data.append(temp_x)
                metrik_name.append(str(counter) + u'. SG or.='+str(p1))
                counter += 1

    # create structure for two axis
    metrik_data1, metrik_data2 = [], []
    x_data1, x_data2 = [], []
    metrik_name1, metrik_name2 = [], []
    for i in xrange(len(metrik_data)):
        if u'' in metrik_name[i] or u'. Average' in metrik_name[i]:
            metrik_data1.append(metrik_data[i])
            x_data1.append(x_data[i])
            metrik_name1.append(metrik_name[i])
        else:
            metrik_data2.append(metrik_data[i])
            x_data2.append(x_data[i])
            metrik_name2.append(metrik_name[i])

    c=0

    fig, ax1 = plt.subplots()
    ax2 = ax1.twinx()
    for i in xrange(len(metrik_data1)):
        y = metrik_data1[i]
        x = x_data1[i]
        ax2.plot(x, y, markers_line[c], label=metrik_name1[i], c=colors[i])
        ax2.text(x[-1], y[-1], metrik_name1[i].split('.')[0],
              horizontalalignment='center',
              bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))
```

```
        c+=1

    for j in xrange(len(metrik_data2)):
        y = metrik_data2[j]
        x = x_data2[j]
        ax1.plot(x, y, markers_line[c], label=metrik_name2[j], c=colors[j+len(metrik_nam
        ax1.text(x[-1], y[-1], metrik_name2[j].split('.')[0],
              horizontalalignment='center',
              bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))
        c+=1

    ax1.set_xlabel(u'Window Width, channels')
    ax1.set_ylabel(u'STD, %')
    ax1.legend(loc='upper left')
    ax2.legend(loc='upper right')
    plt.draw()
    plt.savefig(f+'_spec_smooth_std_eng.png', dpi=300)
    plt.show()
```



In [10]:
```
# max_peak-optimization

# smoothing spectra
test_df = spe_df[spe_df.fraction == 'grain']
ftype_all = Counter(test_df.ftype)
```

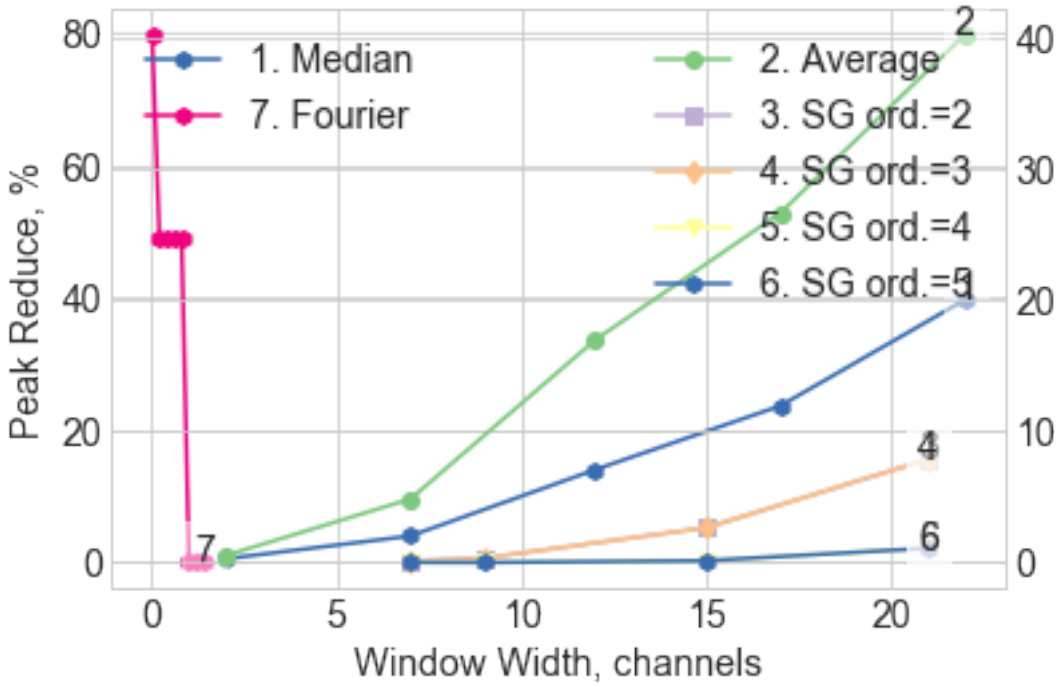
```
        print ftype_all
        aver_spec_type = {}

        smooth_type = {
            0:xrange(2,23,5),
            1:xrange(2,23,5),
            2:[xrange(2,6), (7,9,15,21)],
            3:np.arange(1, 150, 20) / 100.
        }

        data_smooth3 = {}

        for ftype in ftype_all.keys():
            # minimaise delta with average spectra for each type
            test_df2 = test_df[test_df.ftype == ftype]
            spec_all = test_df2.intence
            aver_spec = np.mean(np.array([np.array(x) for x in spec_all]), axis=0)
            aver_spec_type[ftype] = aver_spec
            data_smooth3[ftype] = {}

            # smooth
            for t in smooth_type.keys():
                print ftype, 'smooth type: ', t,
                data_smooth3[ftype][t] = {}
                if t != 2:
                    data_smooth3[ftype][t]['no'] = {}
                    for p in smooth_type[t]:
                        s = []
                        for spec_i in test_df2.intence:
                            s.append(np.max(smooth(spec_i, [p], type_smooth=t)))
                        std_spec_i = np.average(s)
                        data_smooth3[ftype][t]['no'][p] = - (std_spec_i - np.max(aver_spec)) /
                else:
                    data_smooth3[ftype][t] = {}
                    for p in smooth_type[t][0]:
                        data_smooth3[ftype][t][p] = {}
                        for p2 in smooth_type[t][1]:
                            s = []
                            for spec_i in test_df2.intence:
                                s.append(np.max(smooth(spec_i, [p,p2], type_smooth=t)))
                            std_spec_i = np.average(s)
                            data_smooth3[ftype][t][p][p2] = - (std_spec_i - np.max(aver_spec))
                print 'done'
        print 'All Done'
```

```
Counter({'npks': 24})
npks smooth type:  0 done
npks smooth type:  1 done
```

```
npks smooth type:  2 done
npks smooth type:  3 done
All Done
```

```
In [11]: mpl.rcParams.update({'font.size': 14})

         for f in data_smooth3:
             # Create structure for plot
             counter = 1
             metrik_data=[]
             x_data = []
             metrik_name = []
             for a in data_smooth3[f]:
                 if a in [0,1,3]:
                     temp_d = []
                     temp_x = []
                     for p2 in data_smooth3[f][a]['no']:
                         temp_d.append(data_smooth3[f][a]['no'][p2])
                         temp_x.append(p2)
                     temp_x, temp_d = (list(x) for x in zip(*sorted(zip(temp_x, temp_d), key=lam
                     metrik_data.append(temp_d)
                     x_data.append(temp_x)
                     if a == 1:
                         metrik_name.append(str(counter) + u'. Average')
                     elif a == 0:
                         metrik_name.append(str(counter) + u'. Median')
                     else:
                         metrik_name.append(str(counter) + u'. Fourier')
                     counter += 1
                 else:
                     for p1 in data_smooth3[f][a]:
                         temp_d = []
                         temp_x = []
                         for p2 in data_smooth3[f][a][p1]:
                             temp_d.append(data_smooth3[f][a][p1][p2])
                             temp_x.append(p2)
                         temp_x, temp_d = (list(x) for x in zip(*sorted(zip(temp_x, temp_d), key
                         metrik_data.append(temp_d)
                         x_data.append(temp_x)
                         metrik_name.append(str(counter) + u'. SG ord.='+str(p1))
                         counter += 1

             # create structure for two axis
             metrik_data1, metrik_data2 = [], []
             x_data1, x_data2 = [], []
             metrik_name1, metrik_name2 = [], []
             for i in xrange(len(metrik_data)):
```

```
    if u'SG' in metrik_name[i] or u'. Average' in metrik_name[i]:
        metrik_data1.append(metrik_data[i])
        x_data1.append(x_data[i])
        metrik_name1.append(metrik_name[i])
    else:
        metrik_data2.append(metrik_data[i])
        x_data2.append(x_data[i])
        metrik_name2.append(metrik_name[i])

fig, ax1 = plt.subplots()
ax2 = ax1.twinx()
for i in xrange(len(metrik_data1)):
    y = metrik_data1[i]
    x = x_data1[i]
    ax2.plot(x, y, markers_line[i], label=metrik_name1[i], c=colors[i])
    ax2.text(x[-1], y[-1], metrik_name1[i].split('.')[0],
          horizontalalignment='center',
          bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))

for j in xrange(len(metrik_data2)):
    y = metrik_data2[j]
    x = x_data2[j]
    ax1.plot(x, y, markers_line[i], label=metrik_name2[j], c=colors[j+len(metrik_na
    ax1.text(x[-1], y[-1], metrik_name2[j].split('.')[0],
          horizontalalignment='center',
          bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))

ax1.set_xlabel(u'Window Width, channels')
ax1.set_ylabel(u'Peak Reduce, %')
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')
plt.draw()
plt.savefig(f+'_spec_smooth_max_eng.png', dpi=300)
plt.show()
```



In [12]:
```
# shift max_peak-optimization

# smoothing spectra
test_df = spe_df[spe_df.fraction == 'grain']
ftype_all = Counter(test_df.ftype)
print ftype_all
aver_spec_type = {}

smooth_type = {
    0:xrange(2,23,5),
    1:xrange(2,23,5),
    2:[xrange(2,6), (7,9,15,21)],
    3:np.arange(1, 150, 20) / 100.
}

data_smooth4 = {}

for ftype in ftype_all.keys():
    # minimaise delta with average spectra for each type
    test_df2 = test_df[test_df.ftype == ftype]
    spec_all = test_df2.intence
    aver_spec = np.mean(np.array([np.array(x) for x in spec_all]), axis=0)
    aver_spec_type[ftype] = np.where(aver_spec == np.max(aver_spec))[0][0]
    data_smooth4[ftype] = {}
```

```
            # smooth
            for t in smooth_type.keys():
                print ftype, 'smooth type: ', t,
                data_smooth4[ftype][t] = {}
                if t != 2:
                    data_smooth4[ftype][t]['no'] = {}
                    for p in smooth_type[t]:
                        s = []
                        for spec_i in test_df2.intence:
                            ss = smooth(spec_i, [p], type_smooth=t)
                            s.append(np.where(ss == np.max(ss))[0][0])
                        std_spec_i = (aver_spec_type[ftype] - np.abs(np.average(s)))/ aver_spec
                        data_smooth4[ftype][t]['no'][p] = std_spec_i
                else:
                    data_smooth4[ftype][t] = {}
                    for p in smooth_type[t][0]:
                        data_smooth4[ftype][t][p] = {}
                        for p2 in smooth_type[t][1]:
                            s = []
                            for spec_i in test_df2.intence:
                                ss = smooth(spec_i, [p,p2], type_smooth=t)
                                s.append(np.where(ss == np.max(ss))[0][0])
                            std_spec_i = (aver_spec_type[ftype] - np.abs(np.average(s)))/ aver_
                            data_smooth4[ftype][t][p][p2] = std_spec_i
                print 'done'
        print 'All Done'
```

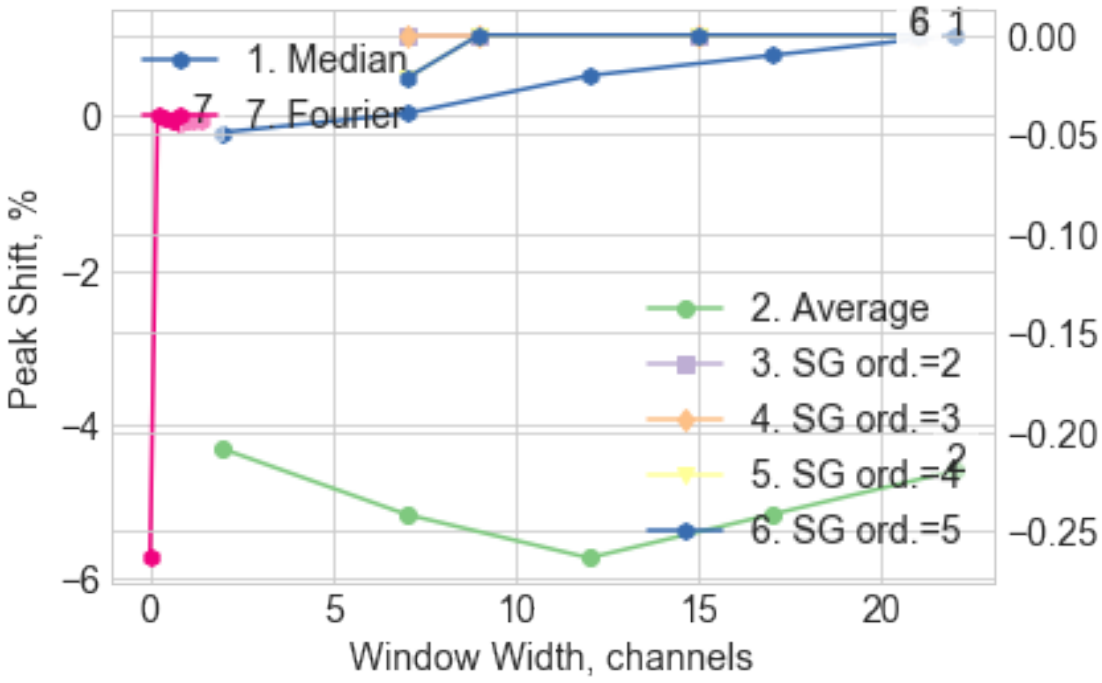
```
Counter({'npks': 24})
npks smooth type:  0 done
npks smooth type:  1 done
npks smooth type:  2 done
npks smooth type:  3 done
All Done
```

```
In [13]: mpl.rcParams.update({'font.size': 14})
```

```
        for f in data_smooth4:
            # Create structure for plot
            counter = 1
            metrik_data=[]
            x_data = []
            metrik_name = []
            for a in data_smooth4[f]:
                if a in [0,1,3]:
                    temp_d = []
                    temp_x = []
```

```
            for p2 in data_smooth4[f][a]['no']:
                temp_d.append(data_smooth4[f][a]['no'][p2])
                temp_x.append(p2)
            temp_x, temp_d = (list(x) for x in zip(*sorted(zip(temp_x, temp_d), key=lam
            metrik_data.append(temp_d)
            x_data.append(temp_x)
            if a == 1:
                metrik_name.append(str(counter) + u'. Average')
            elif a == 0:
                metrik_name.append(str(counter) + u'. Median')
            else:
                metrik_name.append(str(counter) + u'. Fourier')
            counter += 1
        else:
            for p1 in data_smooth4[f][a]:
                temp_d = []
                temp_x = []
                for p2 in data_smooth4[f][a][p1]:
                    temp_d.append(data_smooth4[f][a][p1][p2])
                    temp_x.append(p2)
                temp_x, temp_d = (list(x) for x in zip(*sorted(zip(temp_x, temp_d), key
                metrik_data.append(temp_d)
                x_data.append(temp_x)
                metrik_name.append(str(counter) + u'. SG ord.='+str(p1))
                counter += 1

    # create structure for two axis
    metrik_data1, metrik_data2 = [], []
    x_data1, x_data2 = [], []
    metrik_name1, metrik_name2 = [], []
    for i in xrange(len(metrik_data)):
        if u'SG' in metrik_name[i] or u'. Average' in metrik_name[i]:
            metrik_data1.append(metrik_data[i])
            x_data1.append(x_data[i])
            metrik_name1.append(metrik_name[i])
        else:
            metrik_data2.append(metrik_data[i])
            x_data2.append(x_data[i])
            metrik_name2.append(metrik_name[i])

    fig, ax1 = plt.subplots()
    ax2 = ax1.twinx()
    for i in xrange(len(metrik_data1)):
        y = metrik_data1[i]
        x = x_data1[i]
        ax2.plot(x, y, markers_line[i], label=metrik_name1[i], c=colors[i])
        ax2.text(x[-1], y[-1], metrik_name1[i].split('.')[0],
                 horizontalalignment='center',
```

```
                bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))

        for j in xrange(len(metrik_data2)):
            y = metrik_data2[j]
            x = x_data2[j]
            ax1.plot(x, y, markers_line[i], label=metrik_name2[j], c=colors[j+len(metrik_na
            ax1.text(x[-1], y[-1], metrik_name2[j].split('.')[0],
                  horizontalalignment='center',
                  bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))

        ax1.set_xlabel(u'Window Width, channels')
        ax1.set_ylabel(u'Peak Shift, %')
        ax1.legend(loc='upper left')
        ax2.legend(loc='lower right')
        plt.draw()
        plt.savefig(f+'_spec_smooth_shift_eng.png', dpi=300)
        plt.show()
```



In [14]:
```
# Find of base line

test_df = spe_df[spe_df.fraction == 'grain']
ftype_all = Counter(test_df.ftype)

base_type = {
    1:[[2,6,10,14,20], xrange(5,21,5)],
```

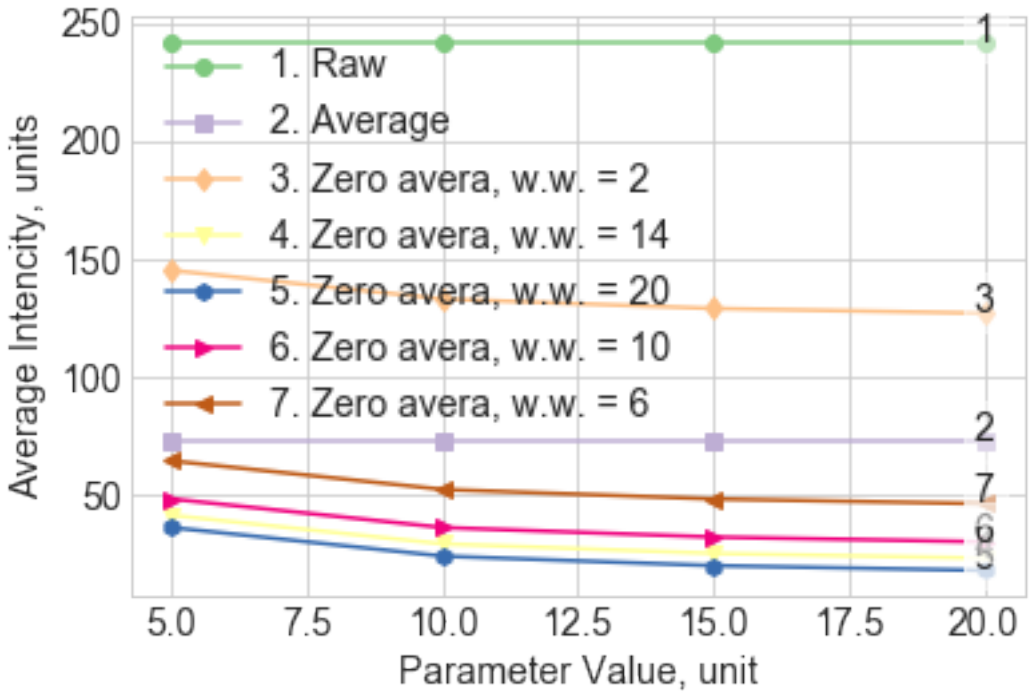
```
    2:xrange(2,40,5),
    3: xrange(10,40,5)  # .. -
}

baseline_spec_type = {}
baseline_spec_aver_int = {}
baseline_spec_aver_len = {}
data_base_max = {}
data_base_len = {}

for ftype in ftype_all.keys():
    # minimaise delta with average spectra for each type
    test_df2 = test_df[test_df.ftype == ftype]
    spec_all = test_df2.intence
    all_spec = np.array([np.array(x) for x in spec_all])
    aver_spec = np.mean(all_spec, axis=0)
    baseline_spec_type[ftype] = aver_spec
    baseline_spec_aver_int[ftype] = np.average(aver_spec[aver_spec < np.mean(aver_spec)
    baseline_spec_aver_len[ftype] = len(aver_spec[aver_spec < np.mean(aver_spec)])
    data_base_max[ftype] = {}
    data_base_len[ftype] = {}
    # find baseline
    for t in base_type.keys():
        print ftype, 'baseline type: ', t,
        data_base_max[ftype][t] = {}
        data_base_len[ftype][t] = {}
        if t != 1:
            data_base_max[ftype][t]['no'] = {}
            data_base_len[ftype][t]['no'] = {}
            for p in base_type[t]:
                s = []
                len_s = []
                for spec_i in test_df2.intence:
                    ss = zero_area(spec_i, [p], type_alg=t)
                    len_s.append(np.average(np.array(spec_i) - np.array(ss)))  # delta
                    s.append(np.average(ss))  # average
                data_base_max[ftype][t]['no'][p] = np.average(s)
                data_base_len[ftype][t]['no'][p] = np.average(len_s)
        else:
            for p in base_type[t][0]:
                print p,
                data_base_max[ftype][t][p]={}
                data_base_len[ftype][t][p]={}
                for p2 in base_type[t][1]:
                    print ' -', p2,
                    s = []
                    len_s = []
                    for spec_i in test_df2.intence:
```

```
                                ss = zero_area(spec_i, [p, p2], type_alg=t)
                                len_s.append(np.average(np.array(spec_i) - np.array(ss)))
                                s.append(np.average(ss))
                            data_base_max[ftype][t][p][p2]=np.average(s)
                            data_base_len[ftype][t][p][p2]=np.average(len_s)
                        print ' +'
                print 'done'
        print 'All Done'
```

```
npks baseline type:  1 2  - 5  - 10  - 15  - 20  +
6  - 5  - 10  - 15  - 20  +
10  - 5  - 10  - 15  - 20  +
14  - 5  - 10  - 15  - 20  +
20  - 5  - 10  - 15  - 20  +
done
npks baseline type:  2 done
npks baseline type:  3

/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:248: RuntimeWarning: invalid value


 done
All Done
```

In [15]:
```
mpl.rcParams.update({'font.size': 14})

        for f in data_base_max:
        # Create structure for plot
            print f
            spec_aver_int = np.mean(baseline_spec_type[f])
            spec_aver_len = len(baseline_spec_type[f])
            sver_filter_int = np.mean(baseline_spec_type[f][baseline_spec_type[f] < np.mean(bas
            sver_filter_len = len(baseline_spec_type[f][baseline_spec_type[f] < np.mean(baselin

            counter = 3
            metrik_data_max = [[spec_aver_int]*4, [sver_filter_int]*4]
            x_data = [range(5,21,5), range(5,21,5)]
            metrik_data_len = [[spec_aver_len]*4, [sver_filter_len]*4]
            metrik_name = [u'1. Raw', u'2. Average']
            for a in data_base_max[f]:
                if a in [2,3]:
                    temp_d_max = []
                    temp_d_len = []
                    temp_x = []
                    for p2 in data_base_max[f][a]['no']:
                        temp_d_max.append(data_base_max[f][a]['no'][p2])
                        temp_d_len.append(data_base_len[f][a]['no'][p2])
                        temp_x.append(p2)
```

```
                    _, temp_d_max = (list(x) for x in zip(*sorted(zip(temp_x, temp_d_max), key=
                    temp_x, temp_d_len = (list(x) for x in zip(*sorted(zip(temp_x, temp_d_len),
                    metrik_data_max.append(temp_d_max)
                    metrik_data_len.append(temp_d_len)
                    x_data.append(temp_x)
                    if a == 2:
                        metrik_name.append(str(counter) + u'. Average Windows')
                    else:
                        metrik_name.append(str(counter) + u'. Parabola')
                    counter += 1
                else:
                    for p1 in data_base_max[f][a]:
                        temp_d_max = []
                        temp_d_len = []
                        temp_x = []
                        for p2 in data_base_max[f][a][p1]:
                            temp_d_max.append(data_base_max[f][a][p1][p2])
                            temp_d_len.append(data_base_len[f][a][p1][p2])
                            temp_x.append(p2)
                        _, temp_d_max = (list(x) for x in zip(*sorted(zip(temp_x, temp_d_max),
                        temp_x, temp_d_len = (list(x) for x in zip(*sorted(zip(temp_x, temp_d_l
                        metrik_data_max.append(temp_d_max)
                        metrik_data_len.append(temp_d_len)
                        x_data.append(temp_x)
                        metrik_name.append(str(counter) + u'. Zero avera, w.w. = ' + str(p1))
                        counter += 1

            fig, ax1 = plt.subplots()

            for i in xrange(len(metrik_data_max)):
                if metrik_name[i].split('.')[0] not in ['8','9']:
                    y = metrik_data_max[i]
                    x = x_data[i]
                    ax1.plot(x, y, markers_line[i], label=metrik_name[i], c=colors[i])
                    ax1.text(x[-1], y[-1], metrik_name[i].split('.')[0],
                          horizontalalignment='center',
                          bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))

            ax1.set_xlabel(u'Parameter Value, unit')
            ax1.set_ylabel(u'Average Intencity, units')
            ax1.legend(loc='upper left')
            plt.draw()
            plt.savefig(f+'_spec_baseline_max_eng.png', dpi=300)
            plt.show()

npks
```



In [16]:
```
#
# Find of base line
test_df = spe_df[spe_df.fraction == 'grain']
ftype_all = Counter(test_df.ftype)

base_type = {
    1:[xrange(0,11,2), xrange(5,26,5)],  # iteration of smooth, threshold
    2:xrange(0,27,2)  # iteration of smooth
    }
data_peaks_num = {}

for ftype in ftype_all.keys():
    # minimaise delta with average spectra for each type
    test_df2 = test_df[test_df.ftype == ftype]
    spec_all = test_df2.intence
    data_peaks_num[ftype] = {}
    # find baseline
    for t in base_type.keys():
        print ftype, 'baseline type: ', t,
        data_peaks_num[ftype][t] = {}
        if t != 1:
            data_peaks_num[ftype][t]['no'] = {}
            for p in base_type[t]:
                s = []
```

```
                        for spec_i in test_df2.intence:
                            auto_find = zero_area(spec_i, params=[10,10], type_alg=1)
                            mask_spec = is_baseline(auto_find) # floor=np.average(auto_find[1:]
                            find_peak = calc_peacs(
                                spec_i,
                                baseline_mask=mask_spec,
                                base_fit=calc_baseline(spec_i, mask_spec),
                                parameters=[p],
                                type_calc=t
                            )[0]
                            s.append(len(find_peak))
                        data_peaks_num[ftype][t]['no'][p] = np.average(s)
                else:
                    for p in base_type[t][0]:
                        print p,
                        data_peaks_num[ftype][t][p]={}
                        for p2 in base_type[t][1]:
                            print ' -', p2,
                            s = []
                            for spec_i in test_df2.intence:
                                auto_find = zero_area(spec_i, params=[10,10], type_alg=1)
                                mask_spec = is_baseline(auto_find) # floor=np.average(auto_find
                                find_peak = calc_peacs(
                                    spec_i,
                                    baseline_mask=mask_spec,
                                    base_fit=calc_baseline(spec_i, mask_spec),
                                    parameters=[p, p2],
                                    type_calc=t
                                )[0]
                                s.append(len(find_peak))
                            data_peaks_num[ftype][t][p][p2]=np.average(s)
                        print ' +'
            print 'done'
    print 'All Done'
```

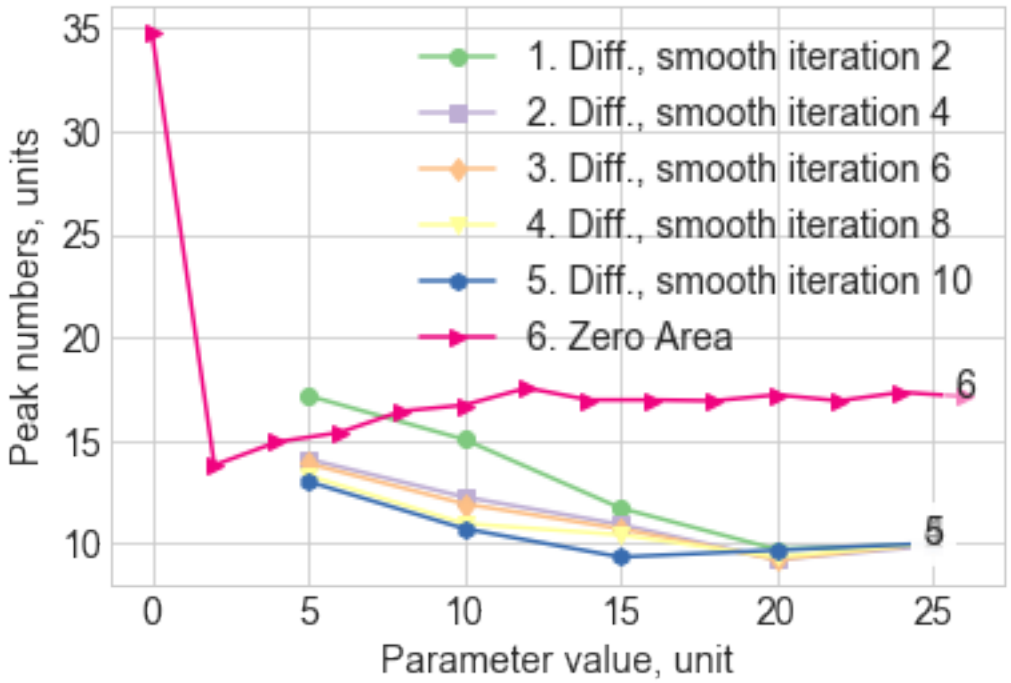
```
npks baseline type:  1 0  - 5

/usr/local/lib/python2.7/dist-packages/numpy/lib/function_base.py:1128: RuntimeWarning: Mean of
  avg = a.mean(axis)
/usr/local/lib/python2.7/dist-packages/numpy/core/_methods.py:80: RuntimeWarning: invalid value
  ret = ret.dtype.type(ret / rcount)


  - 10  - 15  - 20  - 25  +
2  - 5  - 10  - 15  - 20  - 25  +
4  - 5  - 10  - 15  - 20  - 25  +
6  - 5  - 10  - 15  - 20  - 25  +
8  - 5  - 10  - 15  - 20  - 25  +
```

```
10  - 5  - 10  - 15  - 20  - 25  +
done
npks baseline type:  2 done
All Done
```

```
In [17]: mpl.rcParams.update({'font.size': 14})

         for f in data_peaks_num:
         # Create structure for plot
             print f
             counter = 1
             x_data = []
             metrik_data = []
             metrik_name = []
             for a in data_peaks_num[f]:
                 if a != 1:  # zero-area
                     temp_d = []
                     temp_x = []
                     for p2 in data_peaks_num[f][a]['no']:
                         temp_d.append(data_peaks_num[f][a]['no'][p2])
                         temp_x.append(p2)
                     temp_x, temp_d = (list(x) for x in zip(*sorted(zip(temp_x, temp_d), key=lam
                     metrik_data.append(temp_d)
                     x_data.append(temp_x)
                     metrik_name.append(str(counter) + u'. Zero Area')
                     counter += 1
                 else:
                     for p1 in data_peaks_num[f][a]:
                         if max(data_peaks_num[f][a][p1].values()) < 70:
                             temp_d = []
                             temp_x = []
                             for p2 in data_peaks_num[f][a][p1]:
                                 temp_d.append(data_peaks_num[f][a][p1][p2])
                                 temp_x.append(p2)
                             temp_x, temp_d = (list(x) for x in zip(*sorted(zip(temp_x, temp_d),
                             metrik_data.append(temp_d)
                             x_data.append(temp_x)
                             metrik_name.append(str(counter) + u'. Diff., smooth iteration ' + s
                             counter += 1
                         else:
                             pass

             fig, ax1 = plt.subplots()

             for i in xrange(len(metrik_data)):
                 if metrik_name[i].split('.')[0] not in ['8','9']:
                     y = metrik_data[i]
```

```
            x = x_data[i]
            ax1.plot(x, y, markers_line[i], label=metrik_name[i], c=colors[i])
            ax1.text(x[-1], y[-1], metrik_name[i].split('.')[0],
                  horizontalalignment='center',
                  bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))

    ax1.set_xlabel(u'Parameter value, unit')
    ax1.set_ylabel(u'Peak numbers, units')
    ax1.legend(loc='upper right')
    plt.draw()
    plt.savefig(f+'_spec_peaks_eng_v2.png', dpi=300)
    plt.show()
```

npks



1.2

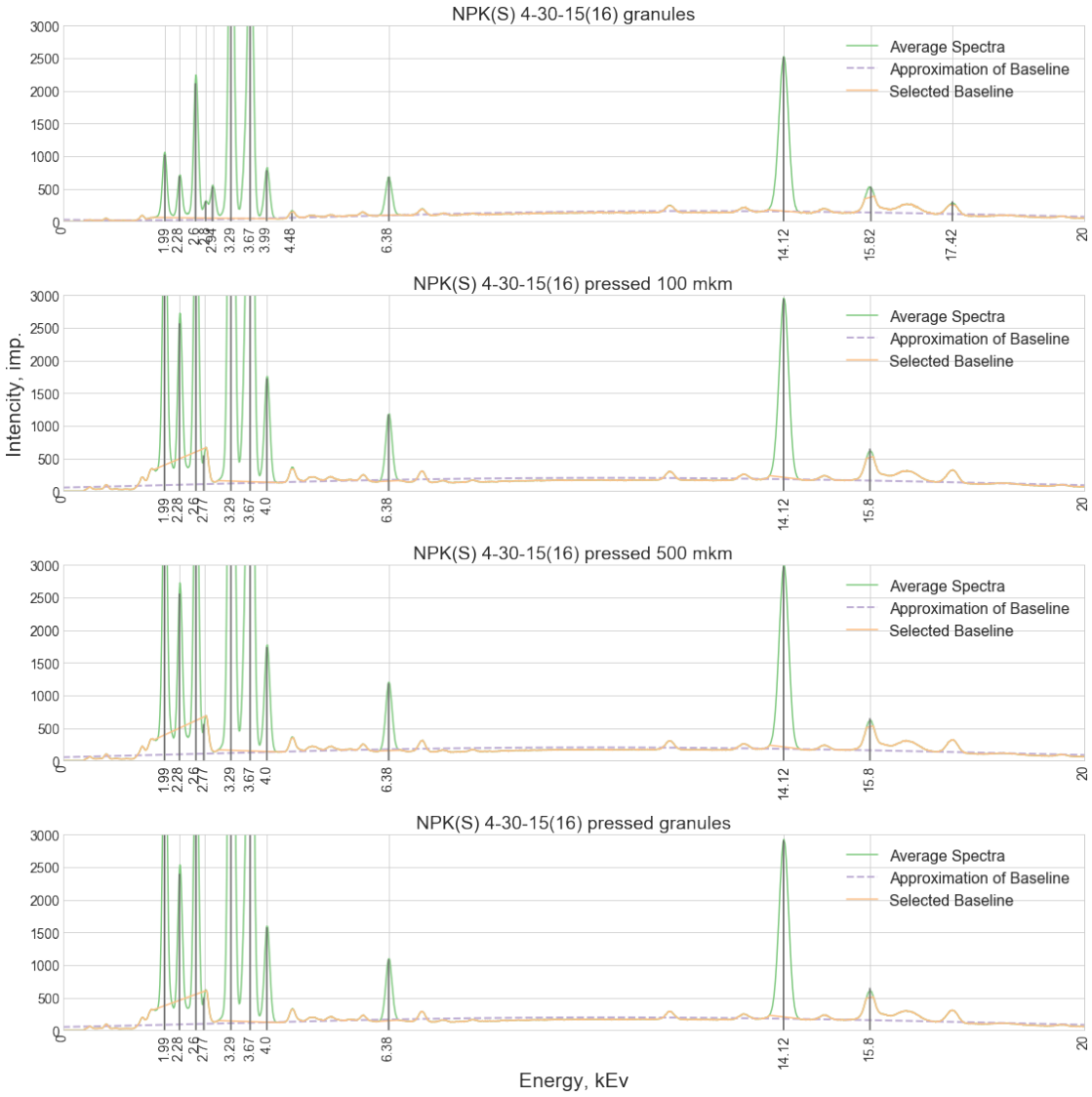
1.3 Work with optimal parameters and save data

In [18]:
```
# Plot data for each type of probe: MAPh, MAPh_kd and NPKS_kd with average on each type
mpl.rcParams.update({'font.size': 16})

spe_df_cl_type = spe_df.copy()
frac_types = list(spe_df_cl_type[spe_df_cl_type.ftype=='npks'].fraction)
```

```
intence = list(spe_df_cl_type[spe_df_cl_type.ftype=='npks'].intence)
marks = ['4.30.15.16']

specs = {}  # {type:[[intenses], [], ...]}
names = []
for i in xrange(len(frac_types)):
    type_obj = 'NPK(S) ' + str(marks[0]) + ' ' + str(frac_types[i])
    if type_obj in specs:
        specs[type_obj].append(intence[i])
    else:
        names.append(type_obj)
        specs[type_obj] = [intence[i]]

aver_specs = {}
for i in specs:
    aver_all = 0
    for j in specs[i]:
        aver_all = aver_all + np.array(j)
    aver_all = aver_all / float(len(specs[i]))
    aver_specs[i] = aver_all

print aver_specs.keys()


aver_specs.pop('NPK(S) 4.30.15.16 100.dry')
aver_specs.pop('NPK(S) 4.30.15.16 500.dry')

reverse_k = {
    'NPK(S) 4.30.15.16 rawgrain': 'NPK(S) 4-30-15(16) granules',
    'NPK(S) 4.30.15.16 grain': 'NPK(S) 4-30-15(16) pressed granules',
    'NPK(S) 4.30.15.16 500': 'NPK(S) 4-30-15(16) pressed 500 mkm',
    'NPK(S) 4.30.15.16 100': 'NPK(S) 4-30-15(16) pressed 100 mkm'
}
aver_specs = dict((reverse_k[key], value) for (key, value) in aver_specs.items())


c = np.arange(len(aver_specs['NPK(S) 4-30-15(16) granules']))
energy = np.average(spe_df_cl_type['a0']) + np.average(spe_df_cl_type['a1'])*c + np.ave

num_plot = len(aver_specs.keys())

fig, ax = plt.subplots(num_plot, figsize=(num_plot*4,num_plot*4))
names_all = list(aver_specs.keys())
print aver_specs.keys()

gauss_h = {}
gauss_s = {}
for i in range(num_plot):
```

```
            name = names_all[i]
            gauss_h[i] = []
            gauss_s[i] = []
            spec = aver_specs[name]
            ax[i].plot(energy, spec, label = u'Average Spectra')

            auto_find = zero_area(spec, params=[14,10], type_alg=1)
            mask_spec = is_baseline(auto_find)  # floor=np.average(auto_find[1:]))
            fit_baseline = calc_baseline(spec_i=spec, baseline_mask=mask_spec)
            ax[i].plot(energy, fit_baseline, '--', label = u'Approximation of Baseline', linewi
            ax[i].plot(energy[mask_spec], spec[mask_spec], label = u'Selected Baseline')
            my_xticks = [0]

            find_peak = calc_peacs(spec, baseline_mask=mask_spec, base_fit=fit_baseline, parame
            for j in range(len(find_peak[1])):
                if find_peak[1][j] < 2500:
                    c = find_peak[1][j]
                    f_e = spe_df['a0'][0] + spe_df['a1'][0]*c + spe_df['a2'][0]*c*c
                    if (round(f_e, 1) not in my_xticks) and (find_peak[0][j] > 2 * find_peak[-1
                        my_xticks.append(round(f_e, 2))
                        gauss_h[i].append(find_peak[2][j])
                        gauss_s[i].append(find_peak[3][j])
                        ax[i].plot([f_e for x in range(int(find_peak[0][j]))],
                                   [x for x in range(int(find_peak[0][j]))], c=colors[-1])

            ax[i].set_xlim((0, 20))
            ax[i].set_ylim((0, 3000))
            ax[i].set_title(names_all[i])
            ax[i].legend(loc = "upper right")
            my_xticks.append(20)
            ax[i].set_xticks(my_xticks)
            ax[i].set_xticklabels(my_xticks, rotation='vertical', fontsize=14)

        ax[-1].set_xlabel(u'Energy, kEv', size=21)
        ax[1].set_ylabel(u'Intencity, imp.', size=21)

        plt.tight_layout()
        plt.draw()
        plt.savefig('aver_spec_npks_fract.png', dpi=300)
        plt.show()
```

```
['NPK(S) 4.30.15.16 500.dry', 'NPK(S) 4.30.15.16 rawgrain', 'NPK(S) 4.30.15.16 100.dry', 'NPK(S)
['NPK(S) 4-30-15(16) granules', 'NPK(S) 4-30-15(16) pressed 100 mkm', 'NPK(S) 4-30-15(16) presse
```



1.4 Prepare all spectra with select algorithm

```
In [19]: names = spe_df_cl_type.index.tolist()
         spe_df_cl_type.loc[:, 'peaks_int']=None
         spe_df_cl_type.loc[:, 'peaks_chan'] = None
         spe_df_cl_type.loc[:, 'gauss_int'] = None
         spe_df_cl_type.loc[:, 'gauss_square'] = None
         spe_df_cl_type.loc[:, 'base_peaks_int'] = None
         spe_df_cl_type.loc[:, 'base_s'] = None
         spe_df_cl_type.loc[:, 'base_max_int'] = None
         spe_df_cl_type.loc[:, 'base_max_chan'] = None
         print 'all data: ', len(names)
```

```
        for j in xrange(len(names)):
            i = names[j]
            spec = spe_df_cl_type.intence[i]
            print j, '. ', i, ' start... ',
            spec = smooth(spec_i=spec, params=[4, 15], type_smooth=2)
            auto_find = zero_area(spec, params=[10,10], type_alg=1)
            mask_spec = is_baseline(auto_find) #floor=np.average(auto_find[1:]))
            fit_baseline = calc_baseline(spec_i=spec, baseline_mask=mask_spec)
            all_peaks_int = []
            all_peaks_chan = []
            all_peak_g_h = []
            all_peak_g_s = []
            all_peak_b = []
            find_peak = calc_peacs(spec, baseline_mask=mask_spec, base_fit=fit_baseline, parame
            for j in xrange(len(find_peak[1])):
                if find_peak[1][j] < 2500 and find_peak[1][j] > 2.*find_peak[-1][j]:
                    all_peaks_int.append(find_peak[0][j])
                    all_peaks_chan.append(int(find_peak[1][j]))
                    all_peak_g_h.append(find_peak[2][j])
                    all_peak_g_s.append(find_peak[3][j])
                    all_peak_b.append(find_peak[4][j])

            spe_df_cl_type.set_value(i, 'peaks_int', all_peaks_int)
            spe_df_cl_type.set_value(i, 'peaks_chan', all_peaks_chan)

            spe_df_cl_type.set_value(i, 'gauss_int', all_peak_g_h)
            spe_df_cl_type.set_value(i, 'gauss_square', all_peak_g_s)
            spe_df_cl_type.set_value(i, 'base_peaks_int', all_peak_b)

            spe_df_cl_type.set_value(i, 'base_s', np.sum(fit_baseline[0:2500]))

            base_max = np.max(fit_baseline[0:2500])
            spe_df_cl_type.set_value(i, 'base_max_int', base_max)
            print 'done'
        print 'All Done'
```

```
all data:  118
0 .  0  start...

/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:35: FutureWarning: set_value is dep
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:36: FutureWarning: set_value is dep
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:38: FutureWarning: set_value is dep
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:39: FutureWarning: set_value is dep
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:40: FutureWarning: set_value is dep
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:42: FutureWarning: set_value is dep
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:45: FutureWarning: set_value is dep
```

```
 done
1 .  1  start...  done
2 .  2  start...  done
3 .  3  start...  done
4 .  4  start...  done
5 .  5  start...  done
6 .  6  start...  done
7 .  7  start...  done
8 .  8  start...  done
9 .  9  start...  done
10 .  10  start...  done
11 .  11  start...  done
12 .  12  start...  done
13 .  13  start...  done
14 .  14  start...  done
15 .  15  start...  done
16 .  16  start...  done
17 .  17  start...  done
18 .  18  start...  done
19 .  19  start...  done
20 .  20  start...  done
21 .  21  start...  done
22 .  22  start...  done
23 .  23  start...  done
24 .  24  start...  done
25 .  25  start...  done
26 .  26  start...  done
27 .  27  start...  done
28 .  28  start...  done
29 .  29  start...  done
30 .  30  start...  done
31 .  31  start...  done
32 .  32  start...  done
33 .  33  start...  done
34 .  34  start...  done
35 .  35  start...  done
36 .  36  start...  done
37 .  37  start...  done
38 .  38  start...  done
39 .  39  start...  done
40 .  40  start...  done
41 .  41  start...  done
42 .  42  start...  done
43 .  43  start...  done
44 .  44  start...  done
45 .  45  start...  done
46 .  46  start...  done
47 .  47  start...  done
```

```
48 .  48  start...  done
49 .  49  start...  done
50 .  50  start...  done
51 .  51  start...  done
52 .  52  start...  done
53 .  53  start...  done
54 .  54  start...  done
55 .  55  start...  done
56 .  56  start...  done
57 .  57  start...  done
58 .  58  start...  done
59 .  59  start...  done
60 .  60  start...  done
61 .  61  start...  done
62 .  62  start...  done
63 .  63  start...  done
64 .  64  start...  done
65 .  65  start...  done
66 .  66  start...  done
67 .  67  start...  done
68 .  68  start...  done
69 .  69  start...  done
70 .  70  start...  done
71 .  71  start...  done
72 .  72  start...  done
73 .  73  start...  done
74 .  74  start...  done
75 .  75  start...  done
76 .  76  start...  done
77 .  77  start...  done
78 .  78  start...  done
79 .  79  start...  done
80 .  80  start...  done
81 .  81  start...  done
82 .  82  start...  done
83 .  83  start...  done
84 .  84  start...  done
85 .  85  start...  done
86 .  86  start...  done
87 .  87  start...  done
88 .  88  start...  done
89 .  89  start...  done
90 .  90  start...  done
91 .  91  start...  done
92 .  92  start...  done
93 .  93  start...  done
94 .  94  start...  done
95 .  95  start...  done
```

```
96 .  96  start...  done
97 .  97  start...  done
98 .  98  start...  done
99 .  99  start...  done
100 .  100  start...  done
101 .  101  start...  done
102 .  102  start...  done
103 .  103  start...  done
104 .  104  start...  done
105 .  105  start...  done
106 .  106  start...  done
107 .  107  start...  done
108 .  108  start...  done
109 .  109  start...  done
110 .  110  start...  done
111 .  111  start...  done
112 .  112  start...  done
113 .  113  start...  done
114 .  114  start...  done
115 .  115  start...  done
116 .  116  start...  done
117 .  117  start...  done
All Done
```

In [20]:
```
# Create elements columns and calculate characteristics line in X-ray spectrum

target_peaks = {
    1.7: 'Si',
    2.0: 'P',
    2.3: 'S',
    2.6: 'Cl',
    3.3: 'K',
    3.7: 'Ca',
    4.5: 'Ti',
    5.9: 'Mn',
    6.4: 'Fe',
    8.1: 'Ta',
    8.6: 'Zn',
    14.2: 'Sr',
    17.4: 'Mo',
    16.5: 'Mo_Coh'
}

ind = spe_df_cl_type.index.tolist()
col_name = target_peaks.keys()

# construct data {col_name: index_value}
```

```
data_all = {}
for e in target_peaks:
    data_all[e] = []
    for i in ind:
        c = int((e - spe_df_cl_type['a0'][i])/spe_df_cl_type['a1'][i])
        d = np.average(spe_df_cl_type['intence'][i][c-1:c+1])
        data_all[e].append(d)


spe_df2 = pd.DataFrame(columns=col_name, index=ind, data=data_all)  # df with channel c


# energy list (equal of channel list) for each index
energy_all = pd.DataFrame(index=ind, data={'energy':[np.average(spe_df_cl_type['a0']) +
            np.average(spe_df_cl_type['a1'])*np.array(x) + \
            np.average(spe_df_cl_type['a2'])*np.array(x)*np.array(x) for x in spe_df_cl

for i in xrange(len(ind)):  # for all index in df
    i_ind = ind[i]  # df index
    data_en = energy_all['energy'][i_ind]
    for en_ind in xrange(len(data_en)):  # for each energy in energy list
        en = round(data_en[en_ind], 1)
        if en in col_name:  # if energy of finding peaks in list of peaks
            spe_df2[en][i_ind] = spe_df_cl_type['peaks_int'][i_ind][en_ind]

new_names_round = [target_peaks[x] for x in spe_df2.columns.tolist()]
spe_df2.columns = new_names_round
print spe_df2.head()


# Gauss int
ind = spe_df_cl_type.index.tolist()
col_name = target_peaks.keys()

def gauss_peaks2(i_all, channel_all, spec_i):
    # calculate Gauss S and I of peaks
    gauss_s = []
    gauss_i = []
    for i in xrange(len(channel_all)):
        c_middle = channel_all[i]
        g_h = 0
        g_s = 0
        w = 15
        y = np.array(spec_i[c_middle-w:c_middle+w])  # gauss fit of points
        x = np.array(xrange(len(y)))
        # correction for weighted arithmetic mean
        mean = np.sum(x * y) / np.sum(y)
        sigma = np.sqrt(np.sum(y * (x - mean)**2) / np.sum(y))
```

```
        try:
            popt,pcov = curve_fit(gaus, x, y, p0=[max(y), mean, sigma])
            y_fit = gaus(x, *popt)
            g_h = np.max(y_fit)
            g_s = g_h * (2 * np.pi) ** 0.5 * popt[-1]  # gauss square
        except RuntimeError as var:
            g_h = 0
            g_s = 0
        except ValueError as var:
            print var
            print x, y
            g_h = 0
            g_s = 0
        gauss_i.append(g_h)
        gauss_s.append(g_s)
    return np.array(gauss_i), np.array(gauss_s)

# construct data {col_name: index_value} with calculate value from raw spectra
data_all = {}
data_all2 = {}
for e in target_peaks:
    data_all[e] = []
    data_all2[e] = []

    for i in ind:
        c = int((e - spe_df_cl_type['a0'][i])/spe_df_cl_type['a1'][i])
        spec = spe_df_cl_type['intence'][i]
        d = gauss_peaks2([spec[c]], [c], spec)
        data_all[e].append(d[0][0])
        data_all2[e].append(d[1][0])


spe_df3 = pd.DataFrame(columns=col_name, index=ind, data=data_all)  # df with channel c
spe_df4 = pd.DataFrame(columns=col_name, index=ind, data=data_all2)  # df with channel

energy_all = pd.DataFrame(index=ind, data={'energy':[np.average(spe_df_cl_type['a0']) +
            np.average(spe_df_cl_type['a1'])*np.array(x) + \
            np.average(spe_df_cl_type['a2'])*np.array(x)*np.array(x) for x in spe_df_cl

for i in xrange(len(ind)):  # for all index in df
    i_ind = ind[i]  # df index
    data_en = energy_all['energy'][i_ind]
    for en_ind in xrange(len(data_en)):  # for each energy in energy list
        en = round(data_en[en_ind], 1)
        if en in col_name:
            spe_df3[en][i_ind] = spe_df_cl_type['gauss_int'][i_ind][en_ind]

new_names_round = ['Gauss.int_' + target_peaks[x] for x in spe_df3.columns.tolist()]
```

```
        spe_df3.columns = new_names_round
        # spe_df3 = spe_df3.loc[:,~spe_df3.columns.duplicated()]

        print spe_df3.head()


        # Gauss square
        for i in xrange(len(ind)):  # for all index in df
            i_ind = ind[i]  # df index
            data_en = energy_all['energy'][i_ind]
            for en_ind in xrange(len(data_en)):  # for each energy in energy list
                en = round(data_en[en_ind], 1)
                if en in col_name:
                    spe_df4[en][i_ind] = spe_df_cl_type['gauss_square'][i_ind][en_ind]

        new_names_round = ['Gauss.sqe_' + target_peaks[x] for x in spe_df4.columns.tolist()]
        spe_df4.columns = new_names_round
        # spe_df3 = spe_df3.loc[:,~spe_df3.columns.duplicated()]

        print spe_df4.head()


        # Concatenate
        _spe_backup = spe_df_cl_type.copy()
        spe_df_cl_type = pd.concat([spe_df_cl_type, spe_df2], axis=1)
        spe_df_cl_type = pd.concat([spe_df_cl_type, spe_df3], axis=1)
        spe_df_cl_type = pd.concat([spe_df_cl_type, spe_df4], axis=1)

        print spe_df_cl_type.shape
        spe_df_cl_type = spe_df_cl_type.loc[:,~spe_df_cl_type.columns.duplicated()]
        print spe_df_cl_type.shape
```

```
           Ti  Mo_Coh            P    Si     Ta     Zn          Fe  \
0  129.384529   272.0  1006.501009  44.0  111.0  101.0  619.249155
1  137.037628   247.5   995.017825  46.0  122.0  105.0  637.019248
2  111.441463   262.0   905.664878  40.0  104.0  109.5  591.678159
3  182.964675   274.5  1040.881217  52.5  120.5  101.5  747.982530
4  143.055810   287.0  1094.369807  49.0  118.0  117.5  730.648810

            S            K          Mo      Sr           Ca     Mn  \
0  671.294053  6031.282690  282.513319  1835.0  4922.477767  140.5
1  649.770450  5944.970328  304.820598  1811.0  4732.075199  119.5
2  677.128804  6254.167335  277.907663  1803.0  4785.358494  110.0
3  734.601407  6654.539539  315.945697  2057.5  5476.197952  158.0
4  758.327250  6515.351330  300.213075  2077.0  5635.215976  146.0

           Cl
0  1940.049791
```

```
1  2056.749509
2  2323.368930
3  2168.094544
4  2001.890261
   Gauss.int_Ti  Gauss.int_Mo_Coh  Gauss.int_P  Gauss.int_Si  Gauss.int_Ta  \
0    143.242691        265.825544  1016.915821           0.0      0.000000
1    146.487391        261.039100  1006.491737           0.0    119.916368
2    138.061561        252.316906   911.764661           0.0    114.079939
3    176.220104          0.000000  1050.789326           0.0      0.000000
4    160.357539        278.652902  1108.443352           0.0    122.784574

   Gauss.int_Zn  Gauss.int_Fe  Gauss.int_S  Gauss.int_K  Gauss.int_Mo  \
0      0.000000    617.833272   684.311373  6205.358513    262.910500
1    114.577794    637.824446   654.506083  6129.051296    273.956953
2    110.590734    598.111303   678.914034  6435.808649    265.395286
3      0.000000    755.271740   746.349434  6854.273488    294.630719
4    120.006867    719.739657   770.405527  6725.137436    289.095685

   Gauss.int_Sr  Gauss.int_Ca  Gauss.int_Mn  Gauss.int_Cl
0   2423.743710   4935.508225    136.599797   2023.531240
1   2395.710670   4743.320123    132.595486   2152.406120
2   2378.925808   4777.895624    126.534640   2430.113743
3   2718.565583   5472.205036    152.124133   2265.172763
4   2680.173291   5652.474968    144.900317   2087.741793
   Gauss.sqe_Ti  Gauss.sqe_Mo_Coh   Gauss.sqe_P  Gauss.sqe_Si  Gauss.sqe_Ta  \
0   3489.316944      19457.642251  14137.471208           0.0      0.000000
1   3537.385776      17659.154834  13797.720353           0.0   8565.473145
2   3295.938904      22075.067717  12513.218346           0.0  14826.830173
3   3857.559135          0.000000  14735.362429           0.0      0.000000
4   3808.544581      24642.611208  15229.513335           0.0   9478.980972

   Gauss.sqe_Zn  Gauss.sqe_Fe   Gauss.sqe_S   Gauss.sqe_K  Gauss.sqe_Mo  \
0      0.000000  12782.834144   9950.664247  86934.671678  10943.905700
1  24134.269235  12718.985814   9686.542485  85417.271458  10321.926453
2  10936.573898  12344.963302  10116.839276  89172.181946  11689.414083
3      0.000000  15003.922805  10903.042334  95391.063701  11901.029557
4  31863.784002  14336.032861  10985.610374  93685.496143  11950.637694

   Gauss.sqe_Sr  Gauss.sqe_Ca  Gauss.sqe_Mn  Gauss.sqe_Cl
0  66125.880105  81113.835877   4909.040539  27369.833763
1  65017.214651  78708.054755   5373.082037  28551.141260
2  65654.177592  79611.013286   4916.262922  32451.903800
3  74405.676539  89784.619791   5521.977542  30136.133442
4  74258.461860  92481.740876   5885.501915  27987.210135
(118, 64)
(118, 64)
```

```
In [21]: spe_df_cl_type.to_pickle('spe_df_cl_type')  # save data frame

In [22]: spe_df_cl_type
```

Out[22]:

	a1	fraction	kd	\
0	0.008923	rawgrain	None	
1	0.008923	rawgrain	None	
2	0.008923	rawgrain	None	
3	0.008923	rawgrain	None	
4	0.008923	rawgrain	None	
5	0.008923	rawgrain	None	
6	0.008923	rawgrain	None	
7	0.008923	rawgrain	None	
8	0.008923	rawgrain	None	
9	0.008923	rawgrain	None	
10	0.008923	grain	None	
11	0.008923	grain	None	
12	0.008923	grain	None	
13	0.008923	grain	None	
14	0.008923	grain	None	
15	0.008923	grain	None	
16	0.008923	grain	None	
17	0.008923	grain	None	
18	0.008923	grain	None	
19	0.008923	grain	None	
20	0.008923	grain	None	
21	0.008923	grain	None	
22	0.008923	grain	None	
23	0.008923	grain	None	
24	0.008923	grain	None	
25	0.008923	grain	None	
26	0.008923	grain	None	
27	0.008923	grain	None	
28	0.008923	grain	None	
29	0.008923	grain	None	
..	...	...	...	
88	0.008923	500	None	
89	0.008923	500.dry	None	
90	0.008923	500.dry	None	
91	0.008923	500.dry	None	
92	0.008923	500.dry	None	
93	0.008923	500.dry	None	
94	0.008923	500.dry	None	
95	0.008923	500.dry	None	
96	0.008923	500.dry	None	
97	0.008923	500.dry	None	
98	0.008923	500.dry	None	
99	0.008923	500.dry	None	

```
100  0.008923   500.dry  None
101  0.008923   500.dry  None
102  0.008923   500.dry  None
103  0.008923   500.dry  None
104  0.008923   100.dry  None
105  0.008923   100.dry  None
106  0.008923   100.dry  None
107  0.008923   100.dry  None
108  0.008923   100.dry  None
109  0.008923   100.dry  None
110  0.008923   100.dry  None
111  0.008923   100.dry  None
112  0.008923   100.dry  None
113  0.008923   100.dry  None
114  0.008923   100.dry  None
115  0.008923   100.dry  None
116  0.008923   100.dry  None
117  0.008923   100.dry  None
```

	energy	spec_num	\
0	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	4	
1	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	4	
2	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	4	
3	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	4	
4	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	4	
5	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	4	
6	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	4	
7	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	4	
8	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	4	
9	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	4	
10	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	14	
11	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	2	
12	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	2	
13	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	2	
14	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	6	
15	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	12	
16	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	10	
17	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	6	
18	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	14	
19	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	12	
20	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	15	
21	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	14	
22	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	10	
23	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	9	
24	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	1	
25	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	6	
26	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	12	
27	[-0.0810641, -0.0721407, -0.0632173, -0.054293...	15	

```
28   [-0.0810641, -0.0721407, -0.0632173, -0.054293...        9
29   [-0.0810641, -0.0721407, -0.0632173, -0.054293...        1
..                                                 ...      ...
88   [-0.0810641, -0.0721407, -0.0632173, -0.054293...        1
89   [-0.0810641, -0.0721407, -0.0632173, -0.054293...        1
90   [-0.0810641, -0.0721407, -0.0632173, -0.054293...        4
91   [-0.0810641, -0.0721407, -0.0632173, -0.054293...        5
92   [-0.0810641, -0.0721407, -0.0632173, -0.054293...        3
93   [-0.0810641, -0.0721407, -0.0632173, -0.054293...        4
94   [-0.0810641, -0.0721407, -0.0632173, -0.054293...        2
95   [-0.0810641, -0.0721407, -0.0632173, -0.054293...        1
96   [-0.0810641, -0.0721407, -0.0632173, -0.054293...        4
97   [-0.0810641, -0.0721407, -0.0632173, -0.054293...        2
98   [-0.0810641, -0.0721407, -0.0632173, -0.054293...        3
99   [-0.0810641, -0.0721407, -0.0632173, -0.054293...        5
100  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        5
101  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        1
102  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        2
103  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        3
104  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        2
105  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        3
106  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        4
107  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        1
108  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        3
109  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        1
110  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        4
111  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        4
112  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        2
113  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        5
114  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        5
115  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        5
116  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        3
117  [-0.0810641, -0.0721407, -0.0632173, -0.054293...        1

                                                  intence        mark  current  \
0    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  4.30.15.16      100
1    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  4.30.15.16      100
2    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  4.30.15.16      100
3    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  4.30.15.16      100
4    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  4.30.15.16      100
5    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  4.30.15.16      100
6    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  4.30.15.16      100
7    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  4.30.15.16      100
8    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  4.30.15.16      100
9    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  4.30.15.16      100
10   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  4.30.15.16      100
11   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  4.30.15.16      100
12   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...  4.30.15.16      100
```

13	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
14	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
15	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
16	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
17	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
18	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
19	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
20	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
21	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
22	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
23	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
24	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
25	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
26	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
27	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
28	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
29	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
..	...	...	...
88	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
89	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
90	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
91	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
92	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
93	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
94	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
95	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
96	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
97	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
98	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
99	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
100	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
101	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
102	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
103	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
104	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
105	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
106	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
107	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
108	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
109	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
110	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
111	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
112	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
113	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
114	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
115	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
116	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100
117	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4.30.15.16	100

	a0	ftype	...	Gauss.sqe_Ta	Gauss.sqe_Zn	Gauss.sqe_Fe	\
0	-0.081064	npks	...	0.000000	0.000000e+00	12782.834144	
1	-0.081064	npks	...	8565.473145	2.413427e+04	12718.985814	
2	-0.081064	npks	...	14826.830173	1.093657e+04	12344.963302	
3	-0.081064	npks	...	0.000000	0.000000e+00	15003.922805	
4	-0.081064	npks	...	9478.980972	3.186378e+04	14336.032861	
5	-0.081064	npks	...	0.000000	0.000000e+00	14644.273729	
6	-0.081064	npks	...	28468.057715	0.000000e+00	15026.333239	
7	-0.081064	npks	...	14330.464954	1.634471e+04	12570.108187	
8	-0.081064	npks	...	16508.984275	0.000000e+00	11836.602490	
9	-0.081064	npks	...	22907.390573	0.000000e+00	13016.098882	
10	-0.081064	npks	...	12168.767084	3.801942e+04	18084.984698	
11	-0.081064	npks	...	33177.564573	2.008907e+04	21999.748065	
12	-0.081064	npks	...	15650.501361	1.249096e+04	18613.290457	
13	-0.081064	npks	...	12785.212189	1.873494e+04	22103.950412	
14	-0.081064	npks	...	10428.422579	1.722302e+04	22684.081868	
15	-0.081064	npks	...	0.000000	1.389143e+04	18922.414174	
16	-0.081064	npks	...	0.000000	0.000000e+00	21301.380619	
17	-0.081064	npks	...	13077.411572	1.512358e+04	23565.440852	
18	-0.081064	npks	...	31868.528409	0.000000e+00	21237.126948	
19	-0.081064	npks	...	11428.295962	1.293526e+04	18322.848533	
20	-0.081064	npks	...	23661.917943	0.000000e+00	22463.688186	
21	-0.081064	npks	...	13378.774225	1.360371e+04	20879.819810	
22	-0.081064	npks	...	31885.211856	1.199114e+04	22285.444182	
23	-0.081064	npks	...	0.000000	3.646227e+04	22089.834582	
24	-0.081064	npks	...	0.000000	2.907814e+04	22729.781269	
25	-0.081064	npks	...	15339.243153	0.000000e+00	24067.769152	
26	-0.081064	npks	...	12983.195261	1.532537e+04	21118.226804	
27	-0.081064	npks	...	0.000000	1.694376e+04	23351.603819	
28	-0.081064	npks	...	22394.064946	0.000000e+00	21708.438724	
29	-0.081064	npks	...	14902.591479	4.196798e+04	21527.987140	
..	...	...	...	...	...	...	
88	-0.081064	npks	...	17459.891209	1.912141e+04	23114.454086	
89	-0.081064	npks	...	11383.449332	2.699227e+04	23974.740146	
90	-0.081064	npks	...	27756.386481	0.000000e+00	22207.420937	
91	-0.081064	npks	...	18034.165889	3.272106e+04	24703.529740	
92	-0.081064	npks	...	37729.405193	1.481763e+04	23667.133694	
93	-0.081064	npks	...	9738.995484	1.032582e+06	22913.562346	
94	-0.081064	npks	...	39778.856377	2.125088e+04	25753.931879	
95	-0.081064	npks	...	11521.894943	1.460006e+04	23345.839991	
96	-0.081064	npks	...	0.000000	0.000000e+00	24356.422465	
97	-0.081064	npks	...	11387.062720	1.673381e+04	26305.593400	
98	-0.081064	npks	...	16603.882690	1.652611e+04	22415.363137	
99	-0.081064	npks	...	13032.133215	1.508637e+04	24848.413533	
100	-0.081064	npks	...	18338.691350	5.217903e+04	24642.968019	
101	-0.081064	npks	...	38781.272262	3.045623e+04	23923.754104	
102	-0.081064	npks	...	16171.824932	0.000000e+00	25702.831983	

```
103 -0.081064  npks      ...        11943.336090  3.155645e+04  22764.527801
104 -0.081064  npks      ...        18682.224660  1.779901e+04  23806.263290
105 -0.081064  npks      ...        14980.769173  2.635840e+04  23248.282644
106 -0.081064  npks      ...        16465.815743  1.928858e+04  23531.119674
107 -0.081064  npks      ...        14689.546031  1.731407e+04  22712.884956
108 -0.081064  npks      ...        16085.003695  0.000000e+00  23080.131174
109 -0.081064  npks      ...       128767.486528  1.221977e+04  23394.874398
110 -0.081064  npks      ...        18470.250894  1.415890e+04  24344.874604
111 -0.081064  npks      ...        13892.623295  0.000000e+00  23354.774451
112 -0.081064  npks      ...        13524.013718  3.322714e+04  23046.616163
113 -0.081064  npks      ...        15836.481220  1.534304e+04  23613.920101
114 -0.081064  npks      ...        13079.930366  0.000000e+00  23206.197311
115 -0.081064  npks      ...        16985.393297  2.972103e+04  23318.744100
116 -0.081064  npks      ...        28068.057352  4.031932e+04  22945.535363
117 -0.081064  npks      ...        23263.906373  0.000000e+00  22988.399475

      Gauss.sqe_S    Gauss.sqe_K  Gauss.sqe_Mo  Gauss.sqe_Sr   Gauss.sqe_Ca  \
0     9950.664247   86934.671678  10943.905700  66125.880105   81113.835877
1     9686.542485   85417.271458  10321.926453  65017.214651   78708.054755
2    10116.839276   89172.181946  11689.414083  65654.177592   79611.013286
3    10903.042334   95391.063701  11901.029557  74405.676539   89784.619791
4    10985.610374   93685.496143  11950.637694  74258.461860   92481.740876
5    10102.485332   92416.008506  11919.901894  72988.272331   87443.420675
6    10173.684196   91750.911548  12281.715791  73525.258635   90682.880967
7     9400.197705   81235.316095  10928.679308  66762.569120   77191.878749
8     9343.140147   83119.955300   9867.151628  63394.499320   72392.560589
9    10533.779147   92094.717260  10863.713674  65003.522088   77511.668721
10   31461.396864  167520.416366  12268.002918  71901.664924  149028.532972
11   34043.224119  176152.435083  13059.701119  77588.835766  165933.128640
12   29767.615193  152361.238746  11635.315627  69148.083974  147868.989290
13   36287.148514  185771.446650  13912.330811  81824.942625  178381.992779
14   42149.038277  211846.018133  14253.788002  89757.119138  173856.056045
15   32948.656074  174243.178389  13054.029077  72548.982421  148822.370711
16   36554.802547  186139.808068  13164.250198  81530.158191  171933.176260
17   37233.406984  190932.863067  12380.715522  88081.186779  169881.641382
18   35993.463377  188605.028636  13101.342033  84474.819191  170201.238709
19   32426.596807  172952.845998  11791.312437  73407.814517  146543.517828
20   36225.292796  189496.419296  13015.032919  81639.403055  180091.003226
21   35112.262728  189916.864769  13295.047631  85000.068670  170866.444743
22   36598.560215  182827.615530  13487.845749  80990.776490  176756.362560
23   33705.061034  167491.996045  12187.912707  80558.920058  163348.523937
24   36841.433201  190034.759929  13696.745871  80858.622457  178694.244484
25   36214.761519  181693.257026  12838.485307  82557.406208  173538.425609
26   36605.342817  194600.250214  14468.045145  81895.421177  169043.320980
27   35996.140988  189018.863597  13651.978890  82163.386277  180632.103208
28   37293.792638  192135.027228  14005.577355  82974.359212  173838.559143
29   36481.904855  190348.936709  13239.906774  78154.024442  174508.646166
..            ...           ...           ...           ...           ...
```

88	38019.188032	208089.944924	13197.902741	81318.882008	187463.126934
89	38107.612934	206724.963254	13671.291461	82092.146036	188299.953087
90	37046.184161	203762.844922	12358.572023	81783.830002	185465.968543
91	37748.446305	204782.863964	15564.227574	81170.486111	187106.810147
92	37977.777617	206006.639031	12770.644638	80635.567466	186516.033975
93	37819.737694	205008.110607	13343.425714	81480.105825	188134.459819
94	37720.562112	203724.862252	14420.529242	80370.124129	184957.368465
95	37941.343840	206511.142515	12932.277570	81273.897915	187187.513234
96	37804.533156	204851.042700	12769.935197	81440.647735	186977.526070
97	37828.757982	205211.894349	12919.082029	80998.038636	186956.824940
98	37963.731304	205931.659146	13445.818063	80612.285504	186840.968117
99	38002.976908	206353.618367	14031.853472	79757.968622	188105.259246
100	37664.123914	206996.811196	13463.996242	81409.568351	187864.300852
101	38148.476564	205136.844090	13240.784584	81245.416868	187329.365544
102	37575.536382	204145.179049	12677.680280	81103.426050	185035.479327
103	37597.435082	207146.384864	12461.215256	81614.548163	187691.783398
104	38029.926437	205609.665196	14119.208047	80518.003477	187605.398882
105	37988.328513	208208.189481	13692.673967	81473.763889	188174.428294
106	38400.279334	204411.657769	15321.356067	79530.716888	186806.840036
107	37866.838929	204752.032768	13838.623040	81455.108594	186801.192919
108	37774.881399	206457.652699	13112.448431	81100.927070	188089.808321
109	37692.590263	206212.742318	14270.444349	81235.710182	187420.734195
110	37810.516746	205109.144999	14845.410236	81066.724362	187987.248988
111	38083.973255	205496.757359	13206.441670	82456.036979	186168.653676
112	38103.534095	206341.081239	13157.511496	81137.200479	187754.208056
113	38228.023308	208221.736640	14758.706129	81751.208621	188845.293994
114	37855.033772	206520.970325	14196.722526	80743.848554	187761.248975
115	38021.713412	206403.039312	14949.336374	80489.275024	187917.469303
116	38259.548801	206990.920068	14032.602396	82260.697460	188132.942594
117	38312.579918	205952.483265	13967.984447	80848.987220	187073.574118

	Gauss.sqe_Mn	Gauss.sqe_Cl
0	4909.040539	27369.833763
1	5373.082037	28551.141260
2	4916.262922	32451.903800
3	5521.977542	30136.133442
4	5885.501915	27987.210135
5	5412.528251	28721.179215
6	5841.398563	25459.707706
7	5129.791153	28071.229452
8	4613.960749	29372.695713
9	4988.685533	34847.103425
10	7359.053110	63792.954162
11	8490.775946	65236.485942
12	6990.959429	53891.413004
13	8524.783568	66098.786523
14	9183.700383	83323.830322
15	8074.282433	65223.889884

```
16   8576.031235  69937.854381
17   8818.906675  71755.672493
18   8782.120916  71812.627351
19   8062.035331  64617.729393
20   8677.966954  68895.354372
21   8984.545843  70510.672079
22   8869.550305  65020.915585
23   8096.842579  58331.152657
24   8674.952322  66198.994369
25   9542.865092  63906.489970
26   9210.756032  71695.627587
27   8415.914766  68536.737534
28   8747.378116  74651.808178
29   8398.805304  70051.554516
..           ...           ...
88   9284.223825  75042.867553
89   9103.238329  74256.929009
90   9211.902317  73404.028755
91   8927.476876  73630.200495
92   8824.445010  73361.674099
93   9069.743840  73740.247513
94   8532.309231  73523.156349
95   9289.213555  74102.178823
96   8883.147261  73661.838849
97   8660.337759  74062.309050
98   9001.481216  73718.790865
99   8791.001603  74428.625300
100  9668.147833  74191.456735
101  8741.891873  73876.645995
102  9329.836744  73821.919521
103  8644.913008  73487.349456
104  8976.735222  71327.791787
105  8806.215639  71653.016004
106  9207.220147  70690.912488
107  9308.656898  71413.879473
108  9094.662508  70988.166469
109  9254.857684  71115.376340
110  8718.233572  70545.709839
111  8692.061631  71241.391159
112  9211.608709  71550.266011
113  9409.090441  71090.017776
114  8876.563170  71423.194114
115  8695.165823  71165.166964
116  9157.021773  71182.171158
117  9125.117111  70681.796694

[118 rows x 64 columns]
```