Multidimensional data fitting tool in Python

Python package for linear and non-linear fitting of multidimensional data.

PySurfaceFit was initially planned as a narrow-scope tool for fitting the molecular potential energy surface (PES) models to the ab initio data computed via quantum chemistry methods. Currently, PySurfaceFit is a flexible general-purpose least squares fitting tool supporting a number of global and local optimization methods.

PySurfaceFit supports fit constraints and Tikhonov regularization for the data points and models parameters. For the Sympy models, based on the Sympy symbolic math library, is is possible to use analytic Jacobians and Hessians if required by the particular optimization method.

Currently, PySurfaceFit heavily relies on the Scipy optimization and root finding library (scipy.optimize).

The basic supported local optimization methods:

1. Levenberg-Marquardt
2. Trust Region Reflective
3. Nelder-Mead
4. Powell
5. CG (Conjugate Gradient)
6. Newton-CG
7. Broyden-Fletcher-Goldfarb-Shanno (BFGS)
8. L-BFGS-B

The basic supported global optimization methods:

1. Simulated Annealing
2. Basin Hopping Optimization

Other custom optimization method backends are fairly easy to implement (see the Notebook tutorials on the optimization methods integration).

Installation

Pulling from Github

1. git clone https://github.com/RomanKochanov/pysurfacefit.git
2. cd pysurfacefit
3. pip install .

From Python Package index

pip install pysurfacefit

PySurfaceFit CLI tool

PySurfaceFit provides two options of usage: as a CLI tool, and as a Python library. The CLI tool provides almost all basic functionality for managing the fit data points, creating fit models, and displaying the results of a fit. For beginners, it is recommended using the CLI tool to avoid most of the programming fuss.

Once installed, PySurfaceFit can be run from the console:

```
pysurfacefit
```

To obtain initial help, use the --help flag:

```
pysurfacefit --help
```

OUTPUT:

```
usage: pysurfacefit [-h] [--config CONFIG] [--startproject STARTPROJECT]
                    [--template TEMPLATE [TEMPLATE ...]] [--merge MERGE]
                    [--set SET] [--init] [--split] [--fit] [--stat]
                    [--codegen] [--plot [PLOT [PLOT ...]]] [--calc]

Python package for linear and non-linear fitting of multidimensional data.

optional arguments:
  -h, --help            show this help message and exit
  --config CONFIG       Configuration file (mandatory for all steps except
                        --startproject)
  --startproject STARTPROJECT
                        Stage 1: start empty project, create dummy config
  --template TEMPLATE [TEMPLATE ...]
                        _________1a: us a template for new project
  --merge MERGE         _________1b: take defaults from a config file
  --set SET             _________1c: explicitly set project parameters
  --init                Stage 2: create model package
  --split               Stage 3: split initial datafile with weighting scheme
  --fit                 Stage 4: start fitting model to data
  --stat                Stage 5: calculate statistics
  --codegen             Stage 6: generate Fortran code for fitted model
  --plot [PLOT [PLOT ...]]
                        Stage 7: plot sections of the model and compare to
                        data
  --calc                Stage 8: calculate model values on grid
```

This will generate an emply project with the sample config file, which will look something like this:

```
####################
# GENERAL SETTINGS #
####################
[GENERAL]

# Name of the project.
project: myproj

##############################
# FITTING DATA SPECIFICATION #
##############################
[DATA]

# CSV file containing fitting data.
datafile:
```

```
# Define rules to split the datafile to dataspec.
# Should be ignored if dataspec is empty.
split_column:
split_values:
split_weights:

# Resulting data specification split-file.
# N.B.: if empty, then the raw datafile is used.
# Data specification *.txt file has the following format:
# alias   path       wht_mul   type    include
# ------- ---------  --------  ------  -------
# test1   test1.csv  1         0       1
# test2   test2.csv  1         0       1
# test3   test3.csv  1         0       1
dataspec: dataspec.txt

# Names and units of the input data columns.
# Input names must be separated by semicolon
# and should not contain dots.
# E.g.: X;Y;Z for names, X:X_UNIT;Y:Y_UNIT for units
input_columns:
input_units:

# Name of the output data column. See explanation above.
output_column:
output_units:

# Weight function.
wht_fun: lambda v: 1

# Global data cutoff
global_cutoff_max:
global_cutoff_min:

################################
# FITTING MODEL SPECIFICATIONS #
################################
[MODEL]

# Model package name.
model: fitmodel

# Model arguments. Argument names must be in the same order
# as the data column names from the DATA section.
# E.g.: X;Y;Z
arguments:

##################
# FITTER OPTIONS #
##################
[FIT]

# Weighted / unweighted fit mode.
weighted_fit: True

# Use "rubber" regularization.
rubber_on: True
```

```
# Fitting method (trf, lm, basinhopping).
# Available "local" methods:
#    "trf" -> Trust Region Reflective (least squares, bounded).
#    "lm"  -> Levenberg Marquardt (least squares, unbounded).
#    'Nelder-Mead'  -> Gradient-free descent Nelder-Mead method.
#    'Powell'  -> modified Powell algorithm.
#    'CG'  -> conjugate gradient algorithm.
#    'BFGS'  -> BFGS algorithm.
#    'Newton-CG'  -> Newton-CG algorithm.
#    'L-BFGS-B'  -> L-BFGS-B algorithm.
#    'TNC'  -> truncated Newton (TNC) algorithm.
#    'COBYLA'  -> Constrained Optimization BY Linear Approximation.
#    'SLSQP'  -> Sequential Least Squares Programming.
#    'trust-constr'  -> minimize a scalar function subject to constraints.
#    'dogleg'  -> dog-leg trust-region algorithm.
#    'trust-ncg'  -> Newton conjugate gradient trust-region algorithm.
#    'trust-exact'  -> "nearly exact trust-region algorithm"
#    'trust-krylov'  -> "early exact trust-region algorithm"
# Available "global" methods:
#    "basinhopping"  -> local descents from a random starting point (bounded)
#    "anneal" -> Simulated Annealing method
fitting_method: trf

# Calculate analytic jacobian (only if ModelSympy is used).
analytic_jacobian: True

# Fit options. Must be given as a list separated by semicolon.
# The valid options for most of the supported fitting methods
# can be found in the corresponding scipy.optimize documentation.
fit_options: max_nfev=200;

##################################
# STATISTICS CALCULATION OPTIONS #
##################################
[STAT]

# Fit statistics file.
stat_file: stat.out

# Calculate outlier statistics.
outlier_stats_flag: True

# Type of statistics: cook, dffits, leverage ,student.
outlier_stats_type: cook

# Output generated symbolic function, for debug purposes only.
output_symbolic_func:

##########################
# CODE GENERATOR OPTIONS #
##########################
[CODEGEN]

# Create Fortran code.
create_fortran: False

# Compare original Python code with the generated Fortran.
```

```
compare_fortran: False

# Fortran compiler executable
compiler_fortran: ifort

# Grid specifications to calculate on.
# Format of the specification must be as follows:
# X=XMIN:XSTEP:XMAX; Y=YMIN:YSTEP:YMAX; ...
gridspec:

####################
# PLOTTING OPTIONS #
####################
[PLOTTING]

# ATTENTION: names of the plot coordinates correspond to the
# model argumen names given in the MODEL section.

# Type of the plot.
# Available plot modes:
#   => "residuals":
#           plot unweighted fit residuals.
#   => "sections":
#           plot sections/cuts of the fitted model vs datapoints.
plot_mode: residuals

# Plot coordinate grid specifications.
# Format of the specification must be as follows:
# X=XVAL; Y=YMIN:YSTEP:YMAX; ...
# In case of the fixed coordinate, variable would only have a value, e.g. X=XVAL.
# In case of unfixed coordinate, there should be a full grid, e.g. Y=YMIN:YSTEP:YMAX
# N.B.: 1) order of the unfixed coords affects the order of the plot axes.
#       2) order of the binding MUST correspond to the argument of the model's
#          __func__ method.
gridspec:

# Model components to plot.
model_components:

# Calculate model's components.
calculate_components: False

# Plot outlier statistics in color. If False, each datagroup has its own color.
plot_outlier_stats: False

# Plot weighted residuals.
resids_weighted: False

# X axes to plot the resuduals versus.
# If empty, defaults to [DATA][output_column].
resids_x_axes:

# Scatter settings (2D, 3D case)
scatter_opacity: 1
marker_size: 20
resize_by_weights: False

# Surface settings (3D case)
```

```
surface_opacity: 0.4

########################
# CALCULATION OPTIONS #
########################
[CALC]

# Output file to output calculated model.
# Column names are defined in DATA section.
output_file: calc.csv

# Grid specifications to calculate on.
# Format of the specification must be as follows:
# X=XMIN:XSTEP:XMAX; Y=YMIN:YSTEP:YMAX; ...
gridspec:
```

Now, let's perform a test fitting case step by step. First, let's generate a three-dimensional data we will fit our model to.

After the test data file is generated, we can create an empty project named "test" from it.

```
pysurfacefit --startproject test
```

To generate a ready-to-go configuration file, one must provide additional data such as names for the argument columns (or "inputs"), and the name of the function column (or "output"). The entered names must correspond to the column names in the data file:

```
Enter the name of the data points CSV file: sampledata.csv
Enter semicolon-separated names for model inputs: x; y; z
Enter name for model output: v
```

If everything went fine, we should have the new fitting project created in a nested subfolder.

```
Created new project: test
New config file has been added: test/config.ini
Sample data specification file has been added: dataspec.txt
```

Now the project folder "test" inlcudes only three files:

```
cd test; ls
config.ini  dataspec.txt  sampledata.csv
```

The file "config.ini" is the main configuration file containing the project settings (see the example of the file above). This file is separated into the following sections:

1. **GENERAL**

Basic settings: contains only the name of the project so far.

2. **DATA**
   Settings for the data to fit: defines the source file for the fitted data, names and units of the input and output columns, weight function and cutoff. Also contains settings for data splitting to apply more advanced weighting schemes which are defined in the dataspec.txt file.

3. **MODEL**
   Containes the model name and names for model arguments. Usually, the argument names are the same as the names of the data inputs defined in the DATA section.

4. **FIT**
   Settings for the fitter. Containes the switches for turning on/off weights and regularization, name of the optimization method, switch for analytic Jacobian, and number of iterations setting.

5. **STAT**
   Additional settings to display the fit statistics.

6. **CODEGEN**
   Settings for the code generation from the fitted model. Allows choosing the compiler. Currently only the Fortan code generation is supported.

7. **PLOTTING**
   Settings for plotting the fit residuals and 1D/2D sections of the fitted model. 3D volume visualization is planned to be included.

8. **CALC**
   Defines the output file and grid to calculate the fited model on.

To fit the data in the sampledata.csv file, let's generate a model file.

```
pysurfacefit --config config.ini --init
```

Note, that for all tasks except --startproject, we must supply the configuraion file using the --config key.

The model is created and stored in a separate Python module file. The name of this file is defined in the MODEL section using the "model" parameter.

```
ls
config.ini  dataspec.txt  fitmodel.py  sampledata.csv
```

The default model has just one fitting parameter which is a constant. Here is what a default model file looks like:

```
import os

from pysurfacefit.models.sympy import ModelSympy
from pysurfacefit.fitpars import Par, Parameters

class fitmodel(ModelSympy):
    """
```

```
        Test model for fitting data with just one parameter.
        """
        def __init__(self,calc_switch='numbified'):
            self.__check_symbolic__ = False
            self.__calc_switch__ = calc_switch # symbolic, lambdified, numbified
            self.__components__ = {}

            # Initialize empty parameters.
            self.__params__ = Parameters()

            # Add constant parameter.
            self.__params__.append(group='constant', pars=[
                Par(name='constant', value=0.0, flag=True),
            ])

        def __units__(self):
            return {"input": {}, "output": "None"}
            def __func__(self,params,x,y,z):

            # constant
            constant = params['constant'].get_value()

            # final result
            res = constant

            # components
            self.__components__['constant'] = constant

            return res

    model = fitmodel()

    if os.path.exists('fitmodel.csv'):
        model.load_params('fitmodel.csv')
    else:
        model.save_params('fitmodel.csv')
```

So far, we will use this default constant model to fit the three-dimensional data we have generated earlier. To do that, use the "fit" command:

```
pysurfacefit --config config.ini --fit
```

This will start the fitting process and will print the intermediate fit statistics to the STDOUT:

```
Treating sampledata fitgroup as FitPoints
BEGIN FIT
USING SCIPY.OPTIMIZE.LEAST_SQUARES: METHOD= trf
METHOD OPTIONS: {'max_nfev': 200, 'bounds': [(-inf,), (inf,)], 'jac': <function Fitter.fit_least_squares.<locals>.
<lambda> at 0x7f84462e54d0>}

==========================================
<<<< calling __sympy_initialize_func__ >>>>
==========================================
Progress:
    - creating sympy objects for inputs
```

```
     - creating sympy objects for parameters
     - get the Sympy expression by calling function with the Sympy objects
     - create lambdified Python function from sympy expression
     - create compiled (numbified) code from the lambdified function
==========================================

CALC FUN     1>>>  DIST:0.000000000e+00  SSE_RUB:0.000000000e+00 SSE_TOT:5.324527156e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.325e+14   UNWHT_SSE:
5.325e+14   WHT_SD: 2.580e+05   UNWHT_SD: 2.580e+05
===============================

==========================================
<<<< calling __sympy_initialize_jac__ >>>>
==========================================
Progress:
     - get the Sympy expression by calling function with the Sympy objects
     - create lambdified Python function from sympy expression
     - create compiled (numbified) code from the lambdified function
==========================================

CALC JAC     1>>>  DIST:0.000000000e+00
CALC FUN     2>>>  DIST:1.000000000e+00  SSE_RUB:0.000000000e+00 SSE_TOT:5.324502669e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.325e+14   UNWHT_SSE:
5.325e+14   WHT_SD: 2.580e+05   UNWHT_SD: 2.580e+05
===============================
CALC JAC     2>>>  DIST:1.000000000e+00
CALC FUN     3>>>  DIST:3.000000000e+00  SSE_RUB:0.000000000e+00 SSE_TOT:5.324453696e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.324e+14   UNWHT_SSE:
5.324e+14   WHT_SD: 2.580e+05   UNWHT_SD: 2.580e+05
===============================
CALC JAC     3>>>  DIST:3.000000000e+00
CALC FUN     4>>>  DIST:7.000000000e+00  SSE_RUB:0.000000000e+00 SSE_TOT:5.324355753e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.324e+14   UNWHT_SSE:
5.324e+14   WHT_SD: 2.580e+05   UNWHT_SD: 2.580e+05
===============================
CALC JAC     4>>>  DIST:7.000000000e+00
CALC FUN     5>>>  DIST:1.500000000e+01  SSE_RUB:0.000000000e+00 SSE_TOT:5.324159873e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.324e+14   UNWHT_SSE:
5.324e+14   WHT_SD: 2.580e+05   UNWHT_SD: 2.580e+05
===============================
CALC JAC     5>>>  DIST:1.500000000e+01
CALC FUN     6>>>  DIST:3.100000000e+01  SSE_RUB:0.000000000e+00 SSE_TOT:5.323768145e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.324e+14   UNWHT_SSE:
5.324e+14   WHT_SD: 2.580e+05   UNWHT_SD: 2.580e+05
===============================
CALC JAC     6>>>  DIST:3.100000000e+01
CALC FUN     7>>>  DIST:6.300000000e+01  SSE_RUB:0.000000000e+00 SSE_TOT:5.322984811e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.323e+14   UNWHT_SSE:
5.323e+14   WHT_SD: 2.579e+05   UNWHT_SD: 2.579e+05
===============================
CALC JAC     7>>>  DIST:6.300000000e+01
CALC FUN     8>>>  DIST:1.270000000e+02  SSE_RUB:0.000000000e+00 SSE_TOT:5.321418635e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.321e+14   UNWHT_SSE:
5.321e+14   WHT_SD: 2.579e+05   UNWHT_SD: 2.579e+05
===============================
CALC JAC     8>>>  DIST:1.270000000e+02
CALC FUN     9>>>  DIST:2.550000000e+02  SSE_RUB:0.000000000e+00 SSE_TOT:5.318288249e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.318e+14   UNWHT_SSE:
5.318e+14   WHT_SD: 2.578e+05   UNWHT_SD: 2.578e+05
===============================
CALC JAC     9>>>  DIST:2.550000000e+02
CALC FUN    10>>>  DIST:5.110000000e+02  SSE_RUB:0.000000000e+00 SSE_TOT:5.312035342e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.312e+14   UNWHT_SSE:
5.312e+14   WHT_SD: 2.577e+05   UNWHT_SD: 2.577e+05
===============================
CALC JAC    10>>>  DIST:5.110000000e+02
CALC FUN    11>>>  DIST:1.023000000e+03  SSE_RUB:0.000000000e+00 SSE_TOT:5.299560985e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.300e+14   UNWHT_SSE:
5.300e+14   WHT_SD: 2.574e+05   UNWHT_SD: 2.574e+05
===============================
CALC JAC    11>>>  DIST:1.023000000e+03
CALC FUN    12>>>  DIST:2.047000000e+03  SSE_RUB:0.000000000e+00 SSE_TOT:5.274738099e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.275e+14   UNWHT_SSE:
5.275e+14   WHT_SD: 2.568e+05   UNWHT_SD: 2.568e+05
===============================
CALC JAC    12>>>  DIST:2.047000000e+03
CALC FUN    13>>>  DIST:4.095000000e+03  SSE_RUB:0.000000000e+00 SSE_TOT:5.225595644e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.226e+14   UNWHT_SSE:
5.226e+14   WHT_SD: 2.556e+05   UNWHT_SD: 2.556e+05
===============================
CALC JAC    13>>>  DIST:4.095000000e+03
CALC FUN    14>>>  DIST:8.191000000e+03  SSE_RUB:0.000000000e+00 SSE_TOT:5.129324001e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.129e+14   UNWHT_SSE:
5.129e+14   WHT_SD: 2.532e+05   UNWHT_SD: 2.532e+05
===============================
CALC JAC    14>>>  DIST:8.191000000e+03
CALC FUN    15>>>  DIST:1.638300000e+04  SSE_RUB:0.000000000e+00 SSE_TOT:4.944833778e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 4.945e+14   UNWHT_SSE:
4.945e+14   WHT_SD: 2.486e+05   UNWHT_SD: 2.486e+05
===============================
CALC JAC    15>>>  DIST:1.638300000e+04
CALC FUN    16>>>  DIST:3.276700000e+04  SSE_RUB:0.000000000e+00 SSE_TOT:4.608065586e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 4.608e+14   UNWHT_SSE:
4.608e+14   WHT_SD: 2.400e+05   UNWHT_SD: 2.400e+05
===============================
CALC JAC    16>>>  DIST:3.276700000e+04
CALC FUN    17>>>  DIST:6.553500000e+04  SSE_RUB:0.000000000e+00 SSE_TOT:4.063378221e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 4.063e+14   UNWHT_SSE:
4.063e+14   WHT_SD: 2.254e+05   UNWHT_SD: 2.254e+05
```

```
===============================
CALC JAC    17>>>  DIST:6.553500000e+04
CALC FUN    18>>>  DIST:1.310710000e+05  SSE_RUB:0.000000000e+00 SSE_TOT:3.489399568e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 3.489e+14   UNWHT_SSE:
3.489e+14   WHT_SD: 2.088e+05   UNWHT_SD: 2.088e+05
===============================
CALC JAC    18>>>  DIST:1.310710000e+05
CALC FUN    19>>>  DIST:1.530418700e+05  SSE_RUB:0.000000000e+00 SSE_TOT:3.450782038e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 3.451e+14   UNWHT_SSE:
3.451e+14   WHT_SD: 2.077e+05   UNWHT_SD: 2.077e+05
===============================
CALC JAC    19>>>  DIST:1.530418700e+05
CALC FUN    20>>>  DIST:1.530418700e+05  SSE_RUB:0.000000000e+00 SSE_TOT:3.450782038e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 3.451e+14   UNWHT_SSE:
3.451e+14   WHT_SD: 2.077e+05   UNWHT_SD: 2.077e+05
===============================
END FIT
9.095534 seconds elapsed, 20 func evals, 19 jac evals
```

As it can be seen from the fit, the result is not good since the generated data is not described by a constaint value. To improve the accuracy of the fit, let's change the code contained in the fitmodel.py by adding the three-dimensional polynomial:

```python
import os
from functools import reduce

from pysurfacefit.models.sympy import ModelSympy
from pysurfacefit.fitpars import Par, Parameters

from pysurfacefit.models.library import generate_powers_layer_3d, poly3d

class fitmodel(ModelSympy):
    """
    Test model for fitting data with just one parameter.
    """
    def __init__(self,calc_switch='numbified'):
        self.__check_symbolic__ = False
        self.__calc_switch__ = calc_switch # symbolic, lambdified, numbified
        self.__components__ = {}

        # Initialize empty parameters.
        self.__params__ = Parameters()

        # Add polynomial parameters
        self.__powers__ = reduce(lambda a,b:a+b,
            [generate_powers_layer_3d(n) for n in range(1,4)]
        )

        for i,j,k in self.__powers__:
            self.__params__.append(group='poly', pars=[
                Par(name='poly_%d_%d_%d'%(i,j,k), value=0.0, flag=True),
            ])

        # Add constant parameter.
        self.__params__.append(group='constant', pars=[
            Par(name='constant', value=0.0, flag=True),
        ])

        def __units__(self):
        return {"input": {}, "output": "None"}

    def __func__(self,params,x,y,z):

        # polynomial
        p = params.get_values(group='poly')
        poly = poly3d(p,x,y,z,self.__powers__)

        # constant
        constant = params['constant'].get_value()

        # final result
        res = poly + constant

        # components
        self.__components__['constant'] = constant

        return res

model = fitmodel()

if os.path.exists('fitmodel.csv'):
    model.load_params('fitmodel.csv')
else:
    model.save_params('fitmodel.csv')
```

Let's run the fit again with more advanced model:

```
pysurfacefit --config config.ini --fit
```

This will start the fitting process and will print the intermediate fit statistics to the STDOUT:

```
Treating sampledata fitgroup as FitPoints
BEGIN FIT
USING SCIPY.OPTIMIZE.LEAST_SQUARES: METHOD= trf
METHOD OPTIONS: {'max_nfev': 200, 'bounds': [(-inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf,
-inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf), (inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf,
inf, inf, inf, inf, inf, inf, inf)], 'jac': <function Fitter.fit_least_squares.<locals>.<lambda> at 0x7f5def410560>}

==========================================
<<<< calling __sympy_initialize_func__ >>>>
==========================================
Progress:
     - creating sympy objects for inputs
     - creating sympy objects for parameters
     - get the Sympy expression by calling function with the Sympy objects
     - create lambdified Python function from sympy expression
     - create compiled (numbified) code from the lambdified function
==========================================

CALC FUN     1>>>  DIST:0.000000000e+00  SSE_RUB:0.000000000e+00 SSE_TOT:5.324527156e+14     ==> WEIGHTED_FIT
```

```
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.325e+14   UNWHT_SSE:
5.325e+14   WHT_SD: 2.580e+05   UNWHT_SD: 2.580e+05
===============================

==========================================
<<<< calling __sympy_initialize_jac__ >>>>
==========================================
Progress:
     - get the Sympy expression by calling function with the Sympy objects
     - create lambdified Python function from sympy expression
     - create compiled (numbified) code from the lambdified function
==========================================

CALC JAC     1>>>  DIST:0.000000000e+00
CALC FUN     2>>>  DIST:1.000000000e+00  SSE_RUB:0.000000000e+00 SSE_TOT:5.223291702e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.223e+14   UNWHT_SSE:
5.223e+14   WHT_SD: 2.555e+05   UNWHT_SD: 2.555e+05
===============================
CALC JAC     2>>>  DIST:1.000000000e+00
CALC FUN     3>>>  DIST:2.999995352e+00  SSE_RUB:0.000000000e+00 SSE_TOT:5.024183860e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.024e+14   UNWHT_SSE:
5.024e+14   WHT_SD: 2.506e+05   UNWHT_SD: 2.506e+05
===============================
CALC JAC     3>>>  DIST:2.999995352e+00
CALC FUN     4>>>  DIST:6.999923483e+00  SSE_RUB:0.000000000e+00 SSE_TOT:4.639397348e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 4.639e+14   UNWHT_SSE:
4.639e+14   WHT_SD: 2.408e+05   UNWHT_SD: 2.408e+05
===============================
CALC JAC     4>>>  DIST:6.999923483e+00
CALC FUN     5>>>  DIST:1.499903800e+01  SSE_RUB:0.000000000e+00 SSE_TOT:3.923332008e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 3.923e+14   UNWHT_SSE:
3.923e+14   WHT_SD: 2.215e+05   UNWHT_SD: 2.215e+05
===============================
CALC JAC     5>>>  DIST:1.499903800e+01
CALC FUN     6>>>  DIST:3.098657439e+01  SSE_RUB:0.000000000e+00 SSE_TOT:2.703038473e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 2.703e+14   UNWHT_SSE:
2.703e+14   WHT_SD: 1.838e+05   UNWHT_SD: 1.838e+05
===============================
CALC JAC     6>>>  DIST:3.098657439e+01
CALC FUN     7>>>  DIST:6.266232927e+01  SSE_RUB:0.000000000e+00 SSE_TOT:1.075696361e+14     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 1.076e+14   UNWHT_SSE:
1.076e+14   WHT_SD: 1.160e+05   UNWHT_SD: 1.160e+05
===============================
CALC JAC     7>>>  DIST:6.266232927e+01
CALC FUN     8>>>  DIST:1.160883053e+02  SSE_RUB:0.000000000e+00 SSE_TOT:1.093359850e+13     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 1.093e+13   UNWHT_SSE:
1.093e+13   WHT_SD: 3.697e+04   UNWHT_SD: 3.697e+04
===============================
CALC JAC     8>>>  DIST:1.160883053e+02
CALC FUN     9>>>  DIST:1.900605403e+02  SSE_RUB:0.000000000e+00 SSE_TOT:2.602526806e+12     ==> WEIGHTED_FIT
```

```
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 2.603e+12   UNWHT_SSE:
2.603e+12   WHT_SD: 1.804e+04   UNWHT_SD: 1.804e+04
===============================
CALC JAC     9>>>  DIST:1.900605403e+02
CALC FUN    10>>>  DIST:3.975523481e+02  SSE_RUB:0.000000000e+00 SSE_TOT:1.507914353e+12     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 1.508e+12   UNWHT_SSE:
1.508e+12   WHT_SD: 1.373e+04   UNWHT_SD: 1.373e+04
===============================
CALC JAC    10>>>  DIST:3.975523481e+02
CALC FUN    11>>>  DIST:8.545961426e+02  SSE_RUB:0.000000000e+00 SSE_TOT:5.250210120e+11     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 5.250e+11   UNWHT_SSE:
5.250e+11   WHT_SD: 8.101e+03   UNWHT_SD: 8.101e+03
===============================
CALC JAC    11>>>  DIST:8.545961426e+02
CALC FUN    12>>>  DIST:1.639588009e+03  SSE_RUB:0.000000000e+00 SSE_TOT:1.170270161e+11     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 1.170e+11   UNWHT_SSE:
1.170e+11   WHT_SD: 3.825e+03   UNWHT_SD: 3.825e+03
===============================
CALC JAC    12>>>  DIST:1.639588009e+03
CALC FUN    13>>>  DIST:3.039288277e+03  SSE_RUB:0.000000000e+00 SSE_TOT:3.211110377e+10     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 3.211e+10   UNWHT_SSE:
3.211e+10   WHT_SD: 2.003e+03   UNWHT_SD: 2.003e+03
===============================
CALC JAC    13>>>  DIST:3.039288277e+03
CALC FUN    14>>>  DIST:6.369248010e+03  SSE_RUB:0.000000000e+00 SSE_TOT:2.288170913e+09     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 2.288e+09   UNWHT_SSE:
2.288e+09   WHT_SD: 5.348e+02   UNWHT_SD: 5.348e+02
===============================
CALC JAC    14>>>  DIST:6.369248010e+03
CALC FUN    15>>>  DIST:7.993910245e+03  SSE_RUB:0.000000000e+00 SSE_TOT:9.174559605e-16     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 9.175e-16   UNWHT_SSE:
9.175e-16   WHT_SD: 3.386e-10   UNWHT_SD: 3.386e-10
===============================
CALC JAC    15>>>  DIST:7.993910245e+03
CALC FUN    16>>>  DIST:7.993910245e+03  SSE_RUB:0.000000000e+00 SSE_TOT:1.549934571e-17     ==> WEIGHTED_FIT
=====data group statistics=====
                    sampledata:   N: 8000   MIN_WHT: 1.0e+00   MAX_WHT: 1.0e+00   WHT_SSE: 1.550e-17   UNWHT_SSE:
1.550e-17   WHT_SD: 4.402e-11   UNWHT_SD: 4.402e-11
===============================
CALC JAC    16>>>  DIST:7.993910245e+03
END FIT
11.802427 seconds elapsed, 16 func evals, 16 jac evals
```

```
/home/roman/work/python/PyDev/PySurfaceFit/git-repo/pysurfacefit-master/showcase/test
jeanny, Ver.3.0
  #  group     names                 values  bounds         weights    flags
---  --------  ----------  ----------------  -----------  ---------  -------
  0  poly      poly_0_0_1   7500             (-inf, inf)          0        1
  1  poly      poly_0_1_0     -2.92517e-11  (-inf, inf)          0        1
  2  poly      poly_1_0_0      1.48667e-12  (-inf, inf)          0        1
  3  poly      poly_0_0_2  -2750             (-inf, inf)          0        1
  4  poly      poly_0_1_1    -3.5252e-13   (-inf, inf)          0        1
  5  poly      poly_0_2_0    -3.17951e-12  (-inf, inf)          0        1
  6  poly      poly_1_0_1    -1.55127e-13  (-inf, inf)          0        1
  7  poly      poly_1_1_0     2.3229e-13   (-inf, inf)          0        1
  8  poly      poly_2_0_0     1            (-inf, inf)          0        1
  9  poly      poly_0_0_3   300            (-inf, inf)          0        1
 10  poly      poly_0_1_2     7.33912e-15  (-inf, inf)          0        1
 11  poly      poly_0_2_1    -1.30851e-14  (-inf, inf)          0        1
 12  poly      poly_0_3_0     0.1          (-inf, inf)          0        1
 13  poly      poly_1_0_2     3.78018e-14  (-inf, inf)          0        1
 14  poly      poly_1_1_1     1.25425e-14  (-inf, inf)          0        1
 15  poly      poly_1_2_0     1.44824e-14  (-inf, inf)          0        1
 16  poly      poly_2_0_1     4.30556e-14  (-inf, inf)          0        1
 17  poly      poly_2_1_0    -1.16635e-14  (-inf, inf)          0        1
 18  poly      poly_3_0_0    -1.22386e-14  (-inf, inf)          0        1
 19  constant  constant      10            (-inf, inf)          0        1
```

The code above lists the fitted parameters for the updated model.