# maxsmooth Documentation

***Release 0.0.0***

**Harry Thomas Jones Bevins**

**Jan 24, 2020**

# CONTENTS:

CHAPTER

ONE

# INTRODUCTION

**maxsmooth** maximally smooth function fitting

**Author** Harry Thomas Jones Bevins

**Version** 0.0.0

**Homepage** https://github.com/htjb/maxsmooth

`maxsmooth` is an open source software for fitting maximally smooth functions ,hearafter MSFs, to data sets. MSFs are functions for which there are no inflection points or, in other words, the high order derivatives do not cross zero within the domain of interest. They are designed to prevent the loss of signals when fitting out dominant foregrounds and in some cases can be used to highlight systematics left in the data.

You can read more about MSFs here ..

`maxsmooth` uses quadratic programming implemented with `cvxopt` to fit data subject to a linear constraint. The constraint on an MSF can be codefied like so,

$$\frac{d^m\,y}{d\,x^m} > 0 \text{ or } \frac{d^m\,y}{d\,x^m} < 0.$$

This constraint is itself not linear but `maxsmooth` is designed to test the constraint,

$$\pm\frac{d^m\,y}{d\,x^m} < 0$$

where a positive sign infront of the $m^{th}$ order derivative forces the derivative to be negative for all x. For an $N^{th}$ order polynomial `maxsmooth` tests every combination of possible signs in front of the derivatives with $m > 2$ for $N <= 10$. For $N > 10$ a smaller subset of the 'sign-space' is tested to reduce runtime but sufficiently large subset to return an accurate fit.

`maxsmooth` features a built in library of maximally smooth functions or allows the user to define their own. The addition of possible inflection points is also available to the user. The software has been designed with these two applications in mind and is a simple interface.

## 1.1 Instalation

## 1.2 Dependencies

Basic requirements:

- Python version..
- pylab

- numpy
- cvxopt

## 1.3 Citation

CHAPTER

TWO

# MAXSMOOTH

## 2.1 Simple Example code

This section is designed to introduce the user to the software and the form in which it is run. In order to run the *maxsmooth* software using the built in MSFs the user can follow the simple structure detailed here.

The user should begin by importing the *setting* class from *maxsmooth.settings* and the *smooth* class from *maxsmooth.msf*.

```
from maxsmooth.settings import setting
from maxsmooth.msf import smooth
```

The *setting* class is used to alter the outputs, model type, fit type, base directory and other attributes of the *smooth* class which is called upon to do the fitting.

The settings of *smooth* should be generated at the start of the code and the attributes changed immediately bellow (see settings for more details).

```
setting = setting()
setting.data_save = True
setting.all_output = True
```

The user should then import the data they wish to fit.

```
import numpy as np

x = np.load('Data/x.npy')
y = np.load('Data/y.npy')
```

and define the polynomial orders they wish to fit as a list.

```
N = [3, 4, 5, 6, 7, 8, 9, 10, 11]
```

or for example,

```
N = [10]
```

*smooth* can be called like so,

```
result = smooth(x, y, N, setting)
```

and it's resulting attributes can be accessed by writing `result.attribute_name`. For example printing the outputs is done like so,

```
print('Objective Funtion Evaluations:\n', result.Optimum_chi)
print('RMS:\n', result.rms)
print('Parameters:\n', result.Optimum_params)
print('Fitted y:\n', result.y_fit)
print('Sign Combinations:\n', result.Optimum_signs)
print('Derivatives:\n', result.derivatives)
```

## 2.2 smooth

*smooth* is used to call the fitting routine by the user.

**class** maxsmooth.msf.**smooth**(*x*, *y*, *N*, *setting*, ***kwargs*)

**Parameters:**

x: *numpy.array* The x data points for the set being fitted.

y: *numpy.array* The y data points for fitting.

N: *list* The number of terms in the MSF polynomial function.

setting: *class atributes* The settings determined by *maxsmooth.settings.setting* and called before smooth().

**Kwargs:**

**initial_params:** ***list of length N*** **Allows the user to overwrite the** the default initial parameters which are a list of length N given by,

```
params0 = [(self.y[-1]-self.y[0])/2]*(self.N)
```

or equivalently in log-space for the 'logarithmic_polynomial' model_type(see Settings),

```
params0 = [(np.log10(self.y[-1])-np.log10(self.y[0]))/2] *
    (self.N)
```

The following Kwargs can be used by the user to define thier own basis function. **Further details on the structures of the following matrix and functions can be found in the section *Designing A Basis Function*. **

**data_matrix:** ***CVXOPT dense matrix of dimensions (len(y),1)*** **The data** matrix is a matrix of y values to be fitted by cvxopt. The default data matrix used by *smooth* is,

```
b = matrix(y, (len(y), 1), 'd').
```

See CVXOPT documentation for details on building a dense matrix. This will only need to be changed on rare occasions when the fitting space is changed. For example smooth will automatically adjust this matrix to,

```
b = matrix(np.log10(y), (len(y), 1), 'd'),
```

when model_type is set to 'logarithmic_polynomial' (see settings).

**basis_function:** ***function with parameters (x, y, mid_point, N)*** **This is** a function of basis functions for the quadratic programming. The variable mid_point is the index at the middle of the datasets x and y.

**model:** ***function with parameters (x, y, mid_point, N, params)*** **This is** a user defined function describing the model to be fitted to the data.

**der_pres:** ***function with parameters (m, i, x, y, mid_point)*** This function describes the prefactors on the ith term of the mth order derivative used in defining the constraint.

**derivatives:** ***function with parameters (m, i, x, y, mid_point, params)*** User defined function describing the ith term of the mth order derivative used to check that conditions are being met.

**args:** ***list*** **of extra arguments for** ***smooth*** **to pass to the functions** detailed above.

**Output**

If N is a list with length greater than 1 then the outputs from smooth are lists and arrays with dimension 0 equal to len(N).

**y_fit:** *numpy.array* The fitted arrays of y data from *smooth*.

**Optimum_chi:** *numpy.array* The optimum chi squared values for the fit calculated by,

$$X^2 = \sum (y - y_{fit})^2.$$

**Optimum_params:** *numpy.array* The set of parameters corresponding to the optimum fits.

**rms:** *list* The rms value of the residuals $y_{res} = y - y_{fit}$ calculated by,

$$rms = \sqrt{\frac{\sum (y - y_{fit})^2}{n}}$$

where $n$ is the number of data points.

**derivatives:** *numpy.array* The $m^{th}$ order derivatives.

**Optimum_signs:** *numpy.array* The sign combinations corresponding to the optimal results.

## 2.3 settings

The Settings class is used to define options that are passed to maxsmooth. It should be called by the user before the function `smooth` by,

```
from maxsmooth.settings import setting
setting = setting()
```

and changes to the settings can be made before a call to `smooth` like so,

```
setting.model_type = 'polynomial'
```

**class** maxsmooth.settings.**setting**

**Attributes**

**fit_type:** (Default==’qp-sign_flipping’)

The type of fitting routine used to fit the model. There are two options designed to explore the sign space of the function.

*Accepted options:*

**‘qp’ - Quadratic programming testing every combination of sign** on the derivatives. This is a quick process provided the order of the polynomial is small.

**‘qp-sign_flipping’ - Quadratic Programming testing a sub** sample of sign combinations on the derivatives. The algorithm currently generates a random set of signs for the $N - 2$ derivatives. It then flips sucessive signs in the list until it calculates a chi squared smaller than the previous evaluation of the objective

function. For example a 4th order polynomial has 2 derivatives with $m >= 2$ which means it has 4 sign combinations [1,1],[-1,-1],[-1,1] and [1,-1]. On first random generation we get [-1,1] with which we evaluate the objective function. We then flip the first sign and evaluate again with [1,1]. If the new chi chi squared is less than the first calculated value the algorithm then goes back to the original list and flips the second sign evaluating with [-1,-1]. The process repeats until the new chi squared is no longer smaller than the previous evaluation and the previous evaluation is taken to be optimal. The algorithm then repeats the entire process for a set number of random sign generations to ensure that the true minimum is identified. The number of repeats needed is dependent on the polynomial order. High polynomial orders require a larger number of repeats to find the true minimum. Currently the number of repeats is set at $2 \times (N-2)^2$.

**model_type:** (Default = 'normalised_polynomial')

The type of model used to fit the data. There is a built in library of maximally smooth functions that can be called by the user.

*Accepted options:*

'normalised_polynomial' - This is a polynomial of the form,

$$y = y_0 \sum (p_i \left( \frac{x}{x_0} \right)^i ).$$

**'polynomial' - This is a polynomial of the form,**

$$y = sum(p_i (x)^i ).$$

**'MSF_2017_polynomial' - This is a polynomial of the form** described in section 4 of Sathyanarayana Rao, 2017

**'logarithmic_polynomial' - This is a polynomial model** similar to that used with the setting 'polynomial' but solved in log-space. It has the form,

$$log_{10}(y) = \sum (p_i (log_{10}(x))^i ).$$

NOTE this model will not work if the y values are negative.

**base_dir: (Default = 'Fitted_Output')** This is the directory in which the output of the program is saved. If the directory does not exist the software will create it in the working as long as the files that preceed it also exist. When testing multiple model types it is recommended to include this in the base directory name eg *self.base_dir= 'Data_Name_' + self.model_type + '/'*.

**cvxopt_maxiter: (Default=1000)** The maximum number of iterations for the cvxopt quadratic programming routine. If cvxopt reaches maxiter the fitting routine will exit with an error recommending this be increased.

**filtering: (Default=True)** Generally for high order N there will be combinations of sign for which CVXOPT cannot find a solution and these terminate with the error "Terminated (Singular KKT Matrix)". If filtering is set to True these cases will be flagged with a warning and the corresponding sign combinations will be excluded when determining the best possible fit. Setting filtering to False will cause the program to crash with CVXOPT error.

**all_output: (Default=False)** If set to True this will output the results of each run of cvxopt to the terminal.

**ifp: (Default = False)** Setting equal to True allows for inflection points in the m order derivatives listed in ifp_derivatives. *NOTE:* The algorithm will not necessarily return derivatives

with inflection points if this is set to True.

*NOTE:* Allowing for inflection points will increasese run time.

**ifp_list: (Default = 'None')** The list of derivatives you wish to allow to have inflection points in(see ifp above). This should be a list of derivative orders eg. if I have a fith order polynomial and I wish to allow the the second derivative to have an inflection point then ifp_list=[2]. If I wished to allow the second and fourth derivative to have inflection points I would write ifp_list=[2,4]. Values in ifp_list cannot exceed the number of possible derivatives and cannot equal 1.

**data_save: (Default = True)** Setting data_save to True will save sample graphs of the derivatives, fit and residuals. The inputs to produce these graphs are all outputted from the *smooth* function and they can be reproduced with more specific axis labels/units in the users code. If filtering is also set to True, which it is by default, then parameters, objective function values and sign combinations from each successful run of cvxopt will be saved to the base directories in seperate folders. The condition on filtering prevents saving data from runs of cvxopt that did not find solutions and terminated with a singular KKT matrix.

## 2.4 Designing A Basis Function

## 2.5 Errors and Warnings

# PYTHON MODULE INDEX

## m

# INDEX

## M



## S