maxsmooth Documentation

Release 0.0.0

Harry Thomas Jones Bevins

Jan 23, 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 smooth

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)
```

2.2 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 and the data sets being fitted are also 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 need be we then go back to the original list and flip the second sign evaluating with [-1,-1]. The process repeats until the new chi squared is no longer smaller than the previous evaluation.(I don't think this process ever repeats more times than there are signs to flip). We then repeat the entire process a set number of times to ensure we can identify the true minimum. 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*(N-2)**2. The sucess of this method can be judged by running with the 'qp' method.

model_type: (Default = 'normalised_polynomial') The type of model used to fit the data. Accepted options:

'normalised_polynomial' - This is a polynomial of the form y=y_0*sum(p_i*(x/x_0)**i). It consistently appears to give the best fit.

'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 doi:10.3847/1538-4357/aa69bd.

'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, for example in the case of uncalibrated data.

'user_defined' - Allows the user to implement their own maximally smooth function to fit to the data.

base_dir: (Default = 'Fitted_Output') This is the directory in which the resultant graphs of the fit, derivatives and residuals are saved. 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.

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)". Setting filtering to True will flag this as a warning and exclude these sign combinations when determining the best possible fit. Setting filtering to False will cause the program to crash with the 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 N-2.

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 msf_fit 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.

2.3 Designing A Basis Function

2.4 Errors and Warnings

PYTHON MODULE INDEX

INDEX