maxsmooth Documentation

Release 1.0.0

Harry Thomas Jones Bevins

Jun 29, 2020

CONTENTS:

CHAPTER

ONE

INTRODUCTION

maxsmooth maximally smooth function fitting

Author Harry Thomas Jones Bevins

Version 1.0.0

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

`maxsmooth` is an open source software for fitting derivative constrained functions, DCFs such as Maximally Smooth Functions , MSFs to data sets. MSFs are functions for which there are no zero crossings in derivatives of order $m \geq 2$ 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. More generally for DCFs the minimum constrained derivative order, m can take on any value or a set of specific high order derivatives can be constrained.

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,

$$\frac{d^m\, y}{d\, x^m} \;\geq\; 0 \text{ or } \frac{d^m\, y}{d\, x^m} \;\leq\; 0.$$

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

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

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` can test every available sign combination but by default it implements a 'sign-smapling' algorithm. This is detailed in the `maxsmooth` paper (see citation) but is summarised below.

The available sign combinations act as discrete parameter spaces all with global minima and `maxsmooth` is capable of finding the minimum of these global minima by implementing a descent algorithm which is followed by a directional exploration. The descent routine typically finds an approximate to the global minimum and then the directional exploration is a complete search of the sign combinations in the neighbourhood of that minimum. The searched region is limited by factors that encapsulate enough of the neighbourhood to confidently return the global minimum.

The sign sampling method is reliant on the problem being 'well defined' but this is not always the case and it is in these instances possible to run the code testing every available sign combination on the constrained derivatives. For a definition of a 'well defined' problem and it's counter part see the `maxsmooth` paper.

`maxsmooth` features a built in library of DCFs or allows the user to define their own. The addition of possible inflection points and zero crossings in higher order derivatives 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 Documentation

The documentation can be compiled from the git repository by...

1.3 Dependencies

Basic requirements:

- Python version..
- matplotlib
- numpy
- cvxopt

1.4 Citation

CHAPTER

TWO

MAXSMOOTH EXAMPLE CODES

This section is designed to introduce the user to the software and the form in which it is run. It provides basic examples of data fitting with a built in MSF model and a user defined model.

There are also examples of functions that can be used pre-fitting and post-fitting for various purposes including; determination of the best DCF model from the built in library for the problem being fitted, analysis of the χ^2 distribution as a function of the discrete sign spaces and analysis of the parameter space surrounding the optimum results.

2.1 Simple Example code

In order to run the `maxsmooth` software using the built in DCF models for a simple fit the user can follow the simple structure detailed here.

The user should begin by importing the *smooth* class from *maxsmooth.DCF*.

```
from maxsmooth.DCF import smooth
```

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.

```
N = [3, 4, 5, 6, 7, 8, 9, 10, 11]
for i in range(len(N)):
    `act on N[i]`
```

or for example,

```
N = 10
```

smooth can be called like so,

```
result = smooth(x, y, N, **kwargs)
```

where the kwargs are detailed below. 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 New Basis Example

2.3 Best Basis Example

2.4 χ^2 Distribution Example

This example will show you how to generate a plot of the χ^2 distribution as a function of the descrete sign combinations on the constrained derivatives.

First you will need to import your data and fit this using `maxsmooth` as was done in the simple example code.

```
import numpy as np

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

from maxsmooth.DCF import smooth

N = 10
result = smooth(x, y, N, base_dir='examples/',
  data_save=True, fit_type='qp')
```

Here we have used some additional keyword arguments for the 'smooth' fitting function. 'data_save' ensures that the files containing the tested sign combinations and the corresponding objective function evaluations exist in the base directory which we have changed to 'base_dir='examples/''. These files are essential for the plotting the χ^2 distribution and are not saved by `maxsmooth` without 'data_save=True'. We have also set the 'fit_type' to 'qp' rather than the default 'qp-sign_flipping'. This ensures that all of the available sign combinations are tested rather than a sampled set giving us a full picture of the distribution when we plot it. We have used the default DCF model to fit this data.

We can import the 'chi_plotter' like so,

```
from maxsmooth.chidist_plotter import chi_plotter
```

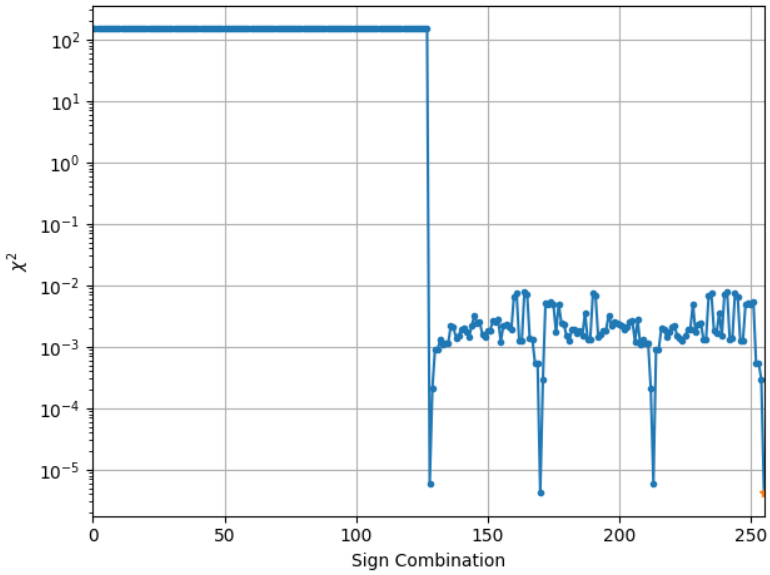
and produce the fit which gets placed in the base directory with the following code,

```
chi_plotter(N, base_dir='examples/', fit_type='qp')
```

We pass the same 'base_dir' as before so that the plotter can find the correct output files. We also give the function the same 'fit_type' used for the fitting which ensures that the files can be read.

The reultant plot is shown below and this can be used to determine how well the sign sampling approach using a descent and directional exploration can find the global minimum. If the distribution looks like noise then it is unlikely the sign sampling algorithm will consistently find the global minimum. Rather it will likely repeatedly return the local minima found after the descent algorithm and you should use the 'qp' method testing all available sign combinations in any future fits to the data with this DCF model.

The distribution for the above problem is shown below where the global minimum is shown as a yellow star.



2.5 Parameter Plotter Example

CHAPTER

THREE

MAXSMOOTH FUNCTIONS

This section details the specifics of the built in functions in `maxsmooth` including the relevant key word arguments and default parameters for all.

3.1 smooth()

smooth is used to call the fitting routine. There are a number of **kwargs that can be assigned to the function which change how the fit is performed, the model that is fit and various other attributes. These are detailed below.

class `maxsmooth.DCF.`**smooth**(*x*, *y*, *N*, ***kwargs*)

Parameters:

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

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

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

Kwargs:

fit_type: Default = 'qp-sign_flipping' This kwarg allows the user to switch between sampling the available discrete sign spaces (default) or testing all sign combinations on the derivatives which can be accessed by setting to 'qp'.

model_type: Default = 'difference_polynomials' Allows the user to access default Derivative Constrained Functions built into the software. Available options include the default, 'polynomial', 'normalised_polynomial', 'legendre', 'log_polynomial', 'loglog_polynomial' and 'exponential'. For more details on the functional form of the built in basis see the `maxsmooth` paper.

pivot_point: Default = len(x)//2 else integer Some of the built in models rely on pivot points in the data sets which by defualt is set as the middle index. This can be altered via this kwarg which can occasionally lead to a better quality fit.

base_dir: Default = 'Fitted_Output/' The location of the outputted data from `maxsmooth`. This must be a string and end in '/'. If the file does not exist then `maxsmooth` will creat it. By default the only outputted data is a summary of the best fit but additional data can be recorded by setting the keyword argument 'data_save = True'.

data_save: Default = False By setting this to True the algorithm will save every tested set of parameters, signs and objective function evaluations into files in base_dir. Theses files will be over written on repeated runs but they are needed to run the 'chidist_plotter'.

all_output: Default = False If set to True this outputs to the terminal every fit performed by the algorithm. By default the only output is the optimal solution once the code is finished.

cvxopt_maxiter: Default = 10000 else integer This shouldn't need changing for most problems however if cvxopt fails with a 'maxiters reached' error message this can be increased. Doing so arbitrarily will however increase the run time of `maxsmooth`.

initial_params: Default = None else list of length N Allows the user to overwrite the default initial parameters used by cvxopt.

constraints: Default = 2 else an integer less than or equal to N - 1 The minimum constrained derivative order which is set by default to 2 for a Maximally Smooth Function.

ifp_list: Default = None else list of integers Allows you to specify if the conditions should be relaxed on any of the derivatives between constraints and the highest order derivative. e.g. a 6th order fit with just a constrained 2nd and 3rd order derivative would have an ifp_list = [4, 5].

cap: Default = (len(available_signs)//N) + N else an integer Determines the maximum number of signs explored either side of the minimum :math`{chi^2}` value found after the decent algorithm has terminated.

chi_squared_limit: Default = 2*min(chi_squared) else float or int The maximum allowed increase in :math`{chi^2}` during the directional exploration. If this value is exceeded then the exploration in one direction is terminated and started in the other. For more details on this and 'cap' see the `maxsmooth` paper.

The following Kwargs can be used by the user to define thier own basis function and will overwrite the 'model_type' kwarg.

basis_function: Default = None else function with parameters (x, y, pivot_point, N) This is a function of basis functions for the quadratic programming. The variable pivot_point is the index at the middle of the datasets x and y by default but can be adjusted.

model: Default = None else function with parameters (x, y, pivot_point, N, params) This is a user defined function describing the model to be fitted to the data.

der_pres: Default = None else function with parameters (m, i, x, y, pivot_point) This function describes the prefactors on the ith term of the mth order derivative used in defining the constraint.

derivatives: Default = None else function with parameters (m, i, x, y, pivot_point, params) User defined function describing the ith term of the mth order derivative used to check that conditions are being met.

args: Default = None else 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.

3.2 best_basis()

3.3 chidist_plotter()

This function allows the user to produce plots of the χ^2 distribution as a function of the available discrete sign spaces for the constrained derivatives. This can be used to identify whether or not the problem is *ill defined*, see the `maxsmooth` paper for a definition, and if it can be solved using the sign sampling approach.

It can also be used to determine whether or not the 'cap' and maximum allowed increase on the value of χ^2 during the directional exploration are sufficient to identify the global minimum for the problem.

The function is reliant on the output of the `maxsmooth` 'smooth' function which can be saved using the 'data_save = True' kwarg when running 'smooth'.

class maxsmooth.chidist_plotter.**chi_plotter**(*N*, ***kwargs*)

3.4 parameter_plotter()

3.5 Errors and Warnings

PYTHON MODULE INDEX

m

INDEX