pwlf: A Python Library for Fitting 1D Continuous Piecewise Linear Functions

Charles F. Jekel*
Gerhard Venter †

January 28, 2019

Name: pwlf
Version: 0.3.2
Description: fit piecewise linear functions to data
License: MIT
Creator: Charles F. Jekel
Maintainer: Charles F. Jekel - cj@jekel.me
Homepage: https://github.com/cjekel/piecewise_linear_fit_py
Contributors: https://github.com/cjekel/piecewise_linear_fit_py/graphs/contributors

Abstract

A Python library to fit continuous piecewise linear functions to one dimensional data is presented. If breakpoint locations are known, a least square fit is used to solve for the best continuous piecewise linear function. Breakpoint locations represent the termination points of the line segments. If breakpoints are unknown, global optimization is used to find the best breakpoint locations by minimizing the sum-of-squares error. This optimization process takes advantage that the least squares fit is a relatively cheap computation for some guess of breakpoint locations. The paper describes the mathematical methods used in the library, provides a brief overview of the library, and presents a few simple examples to illustrate typical use cases.

1 Introduction

Piecewise linear functions are simple functions which consist of several discrete linear pieces that are used to describe a one-dimensional (1D) dependent variable. The locations in which one discrete line ends and a new discrete line begins are referred to as breakpoints. Enforcing continuity between these line segments resulted in a number of interesting models used in a variety of disciplines [1][2][3][4][5][6][7]. Muggeo [8] provides a review of the various techniques that have been used to fit such piecewise continuous models.

*Dept of Mechanical & Aerospace Engineering, University of Florida, Gainesville, FL 32611
†Department of Mechanical and Mechatronic Engineering, Stellenbosch University, Stellenbosch, South Africa

This paper introduces a Python library for fitting continuous piecewise linear functions to 1D data. The library is called *pwlf*, was first available online on April 1, 2017, and includes a number of functions related to fitting continuous piecewise linear models. For instance, the most common use case of *pwlf* is to fit a continuous piecewise linear function for a specified number of line segments. This type of fit is done using global optimization to minimize the sum of squares error between the model and the data. There are two different global optimization strategies included in the library's *fit* and *fitfast* functions. Additionally, a user may specify their own favorite global optimization routine. The library also includes a number of other statistical properties associated with continuous piecewise linear functions.

The paper describes the methodology that *pwlf* uses to fit continuous piecewise linear functions. Essentially a least squares fit can be performed if the breakpoint locations are known [9]. If the breakpoint locations are unknown, then a global optimization routine is used to find the optimal breakpoint locations by minimizing the sum-of-squares error. First, the mathematical methods and various statistical properties available in *pwlf* are presented. Next, a basic overview of the *pwlf* library is provided, followed by a collection of simple examples for typical use cases. The main highlights of the **Contributors:** https://github.com/cjekel/piecewise_linear_fit_py/graphs/contributors library are the following:

- simple Python interface for fitting continuous piecewise linear functions
- fit for specified number of line segments with unknown breakpoint locations
- fit with known breakpoint locations
- constrained fitting that forces model through data points
- quickly predict from a fitted model
- global optimization strategies to find optimal breakpoint locations
- relatively fast and efficient implementation

2 Mathematical formulation

This section describes the mathematical methods used in *pwlf*. Essentially if breakpoints are known, then a solving for the continuous piecewise linear function is a simple linear least squares problem. The linear function can be forced through a set of data points through a constrained least squares formulation. Various statistics associated with the linear regression problem are presented which include the coefficient of determination, standard errors, p-values for model parameters, and the prediction variance of the model. Lastly an optimization problem is presented to find the best breakpoint locations. The optimization problem solves the linear least squares problem several times for various breakpoint combinations by minimizing the sum-of-square of the residual. There is no guarantee that the absolute best (or global minimum) is found, however the methodology has resulted in satisfactory models in a variety of applications.

The first subsection will formulate the least squares problem for fitting a continuous piecewise linear function. The next subsection demonstrates how constraints can be applied to this least squares problem in order to force the continuous piecewise linear function through some set of data points. Various statistics related to continuous piecewise linear functions are then provided. In many practical applications the ideal breakpoint locations are unknown, thus an optimization problem is formulated to find the breakpoint locations that minimize the sum-of-square of the residuals. The optimization strategies utilized in *pwlf* are briefly discussed.

2.1 Least squares with known breakpoints

This work assumes some 1D data set. This work assume that $\mathbf{x}$ is the independent variable and $\mathbf{y}$ is dependent on $\mathbf{x}$ such that $\mathbf{y}(\mathbf{x})$. The data is paired as

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix} \tag{1}$$

where (x_1, y_1) represents the first data point. Additionally, the data points are ordered according to $x_1 \leq x_2 \leq x_3 \leq \cdots \leq x_n$ with n number of data points. A piecewise linear function can be described as the following set of functions

$$\mathbf{y}(x) = \begin{cases} \eta_1 + m_1(x - b_1) & b_1 < x \leq b_2 \\ \eta_2 + m_2(x - b_2) & b_2 < x \leq b_3 \\ \vdots & \vdots \\ \eta_{n_b-1} + m_{n_b-1}(x - b_{n_b-1}) & b_{n_b-1} < x \leq b_{n_b} \end{cases} \tag{2}$$

where b_1 is the x location of the first breakpoint, b_2 is the x location of the second breakpoint, and so forth until the last breakpoint b_{n_b} where there are n_b number of breakpoints. With n_b breakpoints, there are $n_b - 1$ number of line segments. Like the ordering of the data, this formulation also assumes that the breakpoints are ordered as $b_1 < b_2 < \cdots < b_{n_b}$[1].

The above equation represents a set of piecewise linear functions. If it is enforced that the piecewise linear functions are continuous over the domain, then the slopes and intercepts of each linear region become dependent upon previous values. The piecewise functions then reduce to

$$\mathbf{y}(x) = \begin{cases} \beta_1 + \beta_2(x - b_1) & b_1 \leq x \leq b_2 \\ \beta_1 + \beta_2(x - b_1) + \beta_3(x - b_2) & b_2 < x \leq b_3 \\ \vdots & \vdots \\ \beta_1 + \beta_2(x - b_1) + \beta_3(x - b_2) + \cdots + \beta_{n_b}(x - b_{n_b-1}) & b_{n-1} < x \leq b_{n_b} \end{cases} \tag{3}$$

which will result in the same number of unknown β model parameters as the number of breakpoints. These piecewise functions can be expressed in matrix

[1] If the breakpoint locations are not ordered, *pwlf* will automatically order the breakpoints.

form as

$$\begin{bmatrix} 1 & x_1 - b_1 & (x_1 - b_2)\mathbb{1}_{x_1 > b_2} & \cdots & (x_1 - b_{n_b-1})\mathbb{1}_{x_1 > b_{n_b-1}} \\ 1 & x_2 - b_1 & (x_2 - b_2)\mathbb{1}_{x_2 > b_2} & \cdots & (x_2 - b_{n_b-1})\mathbb{1}_{x_2 > b_{n_b-1}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n - b_1 & (x_n - b_2)\mathbb{1}_{x_n > b_2} & \cdots & (x_n - b_{n_b-1})\mathbb{1}_{x_n > b_{n_b-1}} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{n_b} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{4}$$

where $\mathbb{1}_{x_n > b_1}$ represents the indicator function. The indicator functions can be described as piecewise functions that are either 0 or 1, for example

$$\mathbb{1}_{x_n > b_2} = \begin{cases} 0 & x_n \leq b_2 \\ 1 & x_n > b_2 \end{cases} \tag{5}$$

and

$$\mathbb{1}_{x_n > b_3} = \begin{cases} 0 & x_n \leq b_3 \\ 1 & x_n > b_3 \end{cases} \tag{6}$$

and so forth. This is a simple linear system of equations where

$$\boldsymbol{A}\boldsymbol{\beta} = \boldsymbol{y} \tag{7}$$

such that $\mathbf{A}$ is the $n \times n_b$ regression matrix, $\boldsymbol{\beta}$ is the vector $(n_b \times 1)$ of unknown parameters, and $\mathbf{y}$ is the vector $(n \times 1)$ of y data points. The least squares problem solves for the unknown $\boldsymbol{\beta}$ that reduces the sum-of-square of the residuals, and the solution is expressed as

$$\boldsymbol{\beta} = \left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\right)^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}. \tag{8}$$

Once $\boldsymbol{\beta}$ has been solved for, the residual vector is

$$\boldsymbol{e} = \boldsymbol{A}\boldsymbol{\beta} - \boldsymbol{y} \tag{9}$$

where $\boldsymbol{e}$ has the shape of $n \times 1$. The residual vector is the difference between the fitted continuous piecewise linear model and the original data set. The sum-of-squares of the residuals then becomes

$$\mathrm{SSR} = \boldsymbol{e}^{\mathrm{T}}\boldsymbol{e} \tag{10}$$

which is the $L2$ norm of the residual vector.

The regression matrix $\boldsymbol{A}$ has a number of interesting properties. Since the data was ordered initially, $\boldsymbol{A}$ will somewhat resemble a lower triangular matrix[2] with the upper right area of the matrix being filled with zeros. Also, $\boldsymbol{A}$ can be assembled quickly from the ordered x data. This is particularly important when using optimization in cases where the breakpoint locations are unknown, which will require the matrix $\boldsymbol{A}$ to be assembled many times.

[2]Technically a lower triangular matrix will have zeros above the diagonal, and thus $\boldsymbol{A}$ will only be lower triangular for particular data and breakpoint combinations.

2.2 Constrained least squares fit with known breakpoints

In some applications, it may be desirable to force the continuous piecewise function through a particular data point or collection of points. For instance, an unstressed material model must have a stress of zero ($y = 0$) at a strain of zero ($x = 0$). Constrained least squares problems are explained in Chapter 16 of Boyd and Vandenberghe [10]. This subsection extends the least squares problem of fitting continuous piecewise linear functions to a constrained problem when it is desired to force the model through a set of points.

Recall the least squares problem stated in Eqn. 7. It is desired to force the continuous piecewise linear function through

$$\begin{bmatrix} x_{c_1} & y_{c_1} \\ x_{c_2} & y_{c_2} \\ \vdots & \vdots \\ x_{c_{n_c}} & y_{c_{n_c}} \end{bmatrix} \tag{11}$$

with n_c number of constrained data points. Let's call $\boldsymbol{x_c}$ the vector of constrained x locations, and $\boldsymbol{y_c}$ the vector of constrained y locations. A new $c_n \times n_b$ matrix $\boldsymbol{C}$ is assembled where

$$\boldsymbol{C} = \begin{bmatrix} 1 & x_{c_1} - b_1 & (x_{c_1} - b_2)\mathbb{1}_{x_{c_1} > b_2} & \cdots & (x_{c_1} - b_{n_b-1})\mathbb{1}_{x_{c_1} > b_{n_b-1}} \\ 1 & x_{c_2} - b_1 & (x_{c_2} - b_2)\mathbb{1}_{x_{c_2} > b_2} & \cdots & (x_{c_2} - b_{n_b-1})\mathbb{1}_{x_{c_2} > b_{n_b-1}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{c_{n_c}} - b_1 & (x_{c_{n_c}} - b_2)\mathbb{1}_{x_{c_{n_c}} > b_2} & \cdots & (x_{c_{n_c}} - b_{n_b-1})\mathbb{1}_{x_{c_{n_c}} > b_{n_b-1}} \end{bmatrix}. \tag{12}$$

Notice this is the same procedure to assemble $\boldsymbol{A}$, with the exception of using the constrained $\boldsymbol{x_c}$ data instead of $\boldsymbol{x}$.

The Karush–Kuhn–Tucker (KKT) equations for the constrained least squares problem become

$$\begin{bmatrix} 2.0\boldsymbol{A}\mathrm{T}\boldsymbol{A} & \boldsymbol{C}^{\mathrm{T}} \\ \boldsymbol{C} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\beta} \\ \boldsymbol{\zeta} \end{bmatrix} = \begin{bmatrix} 2\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y} \\ \boldsymbol{y_c} \end{bmatrix} \tag{13}$$

where $\boldsymbol{\zeta}$ is some vector of Lagrangian multipliers which will be solved along with the $\boldsymbol{\beta}$ model parameters [10]. This is a square matrix of shape $(n_b + n_c) \times (n_b + n_c)$, and $2\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}$ is a vector with shape $n_b \times 1$. Note that once $\boldsymbol{\beta}$ has been solved, the calculation of the residual vector still follows Eqn. 9.

2.3 Various statistics

Various statistics can be calculated if we assume that the breakpoint locations and model form are correct. This subsection will define the following commonly used regression statistics:

- Coefficient of determination R^2
- Standard error for each $\boldsymbol{\beta}$ model parameter
- Testing for parameter significance with p-values
- Prediction variance due to the uncertainty from the lack of data

2.3.1 Coefficient of determination

The coefficient of determination, commonly refereed to as R^2, compares the correlation between the model output and observed data. First the total sum of squares is calculated using

$$\text{SST} = \sum_{i}^{n} (y_i - \bar{y})^2 \tag{14}$$

where $\bar{y}$ is the mean of $\boldsymbol{y}$. The total sum of squares depends only on the observed data points. Then the coefficient of determination is obtained from

$$R^2 = 1 - \frac{\text{SSR}}{\text{SST}} \tag{15}$$

where SSR is the sum-of-square of the residuals as defined previously in Eqn. 10.

2.3.2 Standard error for each model parameter

The standard error can be calculated for each model parameter. The standard error represents the estimate of the standard deviation of each $\boldsymbol{\beta}$ parameter due to noise in the data. This derivation follows the standard error calculation presented in Coppe et al. [11] for linear regression problems.

First the unbiased estimate of the variance is calculated as

$$\hat{\sigma}^2 = \frac{\text{SSR}}{n - n_b} \tag{16}$$

where n is the number of data points, and n_b is the number of model parameters (or breakpoints used). Then the standard error (SE) for the β_j model parameter is

$$\text{SE}(\beta_j) = \sqrt{\hat{\sigma}^2 \left[\boldsymbol{A}^{\mathrm{T}} \boldsymbol{A}\right]^{-1}_{jj}} \tag{17}$$

for each j parameter ranging from $j = 1, \cdots, n_b$. It is often assumed that the parameters follows a normal distribution, with mean of β_j and standard deviation of $\text{SE}(\beta_j)$.

2.3.3 Test for parameter significance

A statistical test can be done to test for the significance in each $\boldsymbol{\beta}$ model parameter. This is a marginal test as defined in section 2.4.2 of [12], because the $\boldsymbol{\beta}$ parameters are codependent on each other.

The hypotheses for testing the significance of any individual regression parameter β_j are

$$H_0 : \beta_j = 0 \tag{18}$$

and

$$H_1 : \beta_j \neq 0. \tag{19}$$

If H_0 is not rejected, then β_j may be deleted from the model. This could imply that too many line segments are being used for the provided data. The test statistic is

$$t_j = \frac{\beta_j}{\text{SE}(\beta_j)} \tag{20}$$

or the ratio of the parameter to it's standard error. The p-value for each parameters is the probability of obtaining a test statistic greater than $|t_j|$. Note that t_j follows Student's t-distribution, with $(n - nb - 1)$ degrees of freedom. A typically hypothesis test would reject H_0 if the p-value is greater than some level of significance α.

2.3.4 Prediction variance due to the uncertainty from the lack of data

The prediction variance is a useful tool for assessing the model uncertainty. For continuous piecewise linear functions, the prediction variance represents the uncertainty in each linear segment due to the lack of data in that regime. For a useful discussion on the prediction variance, refer to section 8.4.4 of [12].

The regression matrix shall be denoted $\hat{\boldsymbol{A}}$ when it is assembled on a set of new prediction points $\hat{\boldsymbol{x}}$, and the predictive model output is given as $\hat{\boldsymbol{y}} = \hat{\boldsymbol{A}}\boldsymbol{\beta}$. Note that $\hat{\boldsymbol{x}}$ can contain any number of data points from the original domain of $\boldsymbol{x}$. The reason for this is that the prediction variance can be calculated at any x within the model, and is not restricted to the original $\boldsymbol{x}$ data. The prediction variance as a function of $\hat{\boldsymbol{x}}$ is given by

$$\mathrm{PV}(\hat{\boldsymbol{x}}) = \hat{\sigma}^2 \mathrm{diag}\left(\hat{\boldsymbol{A}} [\boldsymbol{A}^{\mathrm{T}} \boldsymbol{A}]^{-1} \hat{\boldsymbol{A}}^{\mathrm{T}} \right) \tag{21}$$

where diag represents the diagonal along the matrix. It's generally assumed that $\hat{\boldsymbol{y}}$ follows a normal distribution, with standard deviation equal to $\sqrt{\mathrm{PV}(\hat{\boldsymbol{x}})}$.

2.4 Finding the optimal breakpoints

The fitting of continuous piecewise linear functions has thus far assumed that the breakpoint locations are known. In cases when the breakpoint locations are unknown, optimization is used to find the best set of breakpoint locations. All that is needed in *pwlf* is for the user to specify the desired number of line segments. Remember there are $nb - 1$ number of line segments.

For any given set of breakpoint locations $\boldsymbol{b}$, a least squares fit can be performed which solves for the $\boldsymbol{\beta}$ parameters that minimize the sum-of-square of the residuals. The sum-of-square of the residuals can be represented as a function of the breakpoint location $\mathrm{SSR}(\boldsymbol{b})$. In an effort to reduce the number of variables in the optimization problem, *pwlf* always assumes that the first breakpoint is $b_1 = x_1$ (or the smallest x), and the last breakpoint is $b_{n_b} = x_n$ (or the largest x). An optimization problem is formulated to find the breakpoint locations that minimize the overall sum-of-square of the residuals. The summary of the optimization problem is as follows:

$$\begin{aligned} \text{minimize} \quad & \mathrm{SSR}(\boldsymbol{b}), \quad \boldsymbol{b} = [b_2, \cdots, b_{n_b - 1}]^T \\ \text{subject to} \quad & x_1 \leq b_k \leq x_n, \quad k = 1, 2, \cdots, n_b. \end{aligned}$$

Two different optimization strategies are currently utilized. The first strategy utilizes the Differential Evolution (DE) global optimization strategy [13]. The DE optimization algorithm is used in the *fit* function as *pwlf*'s default optimization strategy. The specific DE strategy being used is the SciPy DE implementation [14]. The DE optimization strategy is a very popular heuristic

optimizer that has been used to solve a variety of problems, and is well suited to find the global optimum provided a very large number of function evaluations. However, cases may arise where the progress of DE is deemed too slow, too expensive, or the DE result is consistently undesirable as there is no guarantee that DE will find the global optimum given any fixed number of function evaluations.

An alternative multi-start gradient based optimization strategy is used in the *fitfast* function for cases where DE may be undesirable. The multi-start optimization algorithim first randomly selects an initial population of starting points from a latin-hypercube sampling[3], which is a space filling design of experiments. Each starting point is a unique combination of random breakpoint locations. From each starting point, a local optimization algorithm is run which minimizes the sum-of-square errors. Running multiple local optimiizaztions is a strategy that attempts to find the global optimum, and such a strategy was mentioned by Muggeo [15] for solving problems with multiple local minima. The local optimization algorithm being used is the LBFGS [16] gradient based optimizer implmented in SciPy [14]. Schutte et al. [17] observed a case where the multi-start optimization performance may exceed running one global optimization algorithm for an extended period of time. The caveat with the multi-start local optimization algorithms is that each individual optimization may get stuck at a local minima. Increasing the number of starting points will increase the chances of finding the global optimum.

The described optimization process is an optimization within an optimization, or a double-loop optimization. There is an inner optimization occurring at every function evaluation of the overall process. This inner optimization minimizes the sum-of-squares to find the best continuous piecewise linear model parameters for a given set of breakpoint locations. It will be required to solve the least squares problem several times within the outer optimization process. As shown latter in the examples, the outer optimization process can be used to find breakpoint locations in a short amount of time on a modern computer. This is largely possible because of how efficient the inner loop is at finding model parameters using the least squares method. Finding the breakpoint locations for other objective functions (i.e. minimizing absolute average deviation, or any other LN norm) would be a considerably more expensive problem, because an iterative solver would be required within the inner optimization process.

3 The pwlf Python library

A brief overview of the *pwlf* library is provided. This information includes installation details, versioning semantics, and details about the fitting class.

3.1 Installation

It is recommended to install *pwlf* using pip by running

```
pip install pwlf
```

[3]The latin-hypercube sampling is done using the pyDOE package. https://pythonhosted.org/pyDOE/

in a shell (or command prompt)[4]. This will download and install the latest *pwlf* release along with the necessary dependencies. The dependencies are the following:

- Python >= 2.7
- NumPy >= 1.14.0
- SciPy >= 0.19.0
- pyDOE >= 0.3.8

Alternatively, *pwlf* can be installed from the source code by running the following code.

```
git clone https://github.com/cjekel/piecewise_linear_fit_py.git
pip install ./piecewise_linear_fit_py
```

3.2 Versioning

To import and check the version of *pwlf* run

```
import pwlf
pwlf.__version__
```

where *pwlf.__version__* is a string following "MAJOR.MINOR.PATCH" Semantic Versioning[5]. The changelog is hosted online at https://github.com/cjekel/piecewise_linear_fit_py/blob/master/CHANGELOG.md, and released versions of *pwlf* are available online at https://pypi.org/project/pwlf/. A new release will be uploaded for changes in the source code, however the most minor changes (typos in docstrings or example files) may not be released to PyPI.org.

Run the following code to upgrade to the latest version.

```
pip install pwlf --upgrade --no-deps
```

3.3 PiecewiseLinFit class

The useage of *pwlf* was largely inspired by the simplicity of the scikit-learn project [18], in which the entire fitting routine is stored inside the *PiecewiseLinFit* class. The object is initialized by calling

```
model = PiecewiseLinFit(x, y, disp_res=False, sorted_data=False)
```

[4]The PyPA recommended tool for installing packages is with pip https://pypi.org/project/pip/

[5]https://semver.org/spec/v2.0.0.html

where *model* becomes the working object in which all fitting routines are run for the particular x and y data. If the breakpoint locations are known, use *model.fit_with_breaks* to perform a least squares fit. If breakpoint locations are unknown, use *model.fit* and *model.fitfast* to perform a fit by specifying the desired number of line segments. Once a fit has been performed, the object will contain the following attributes:

```
model.ssr  # sum of squares error
model.fit_breaks  # breakpoint locations
model.n_parameters  # number of model parameters
model.n_segments  # number of line segments
model.beta  # model parameters
model.slopes  # slope of each line segment
model.intercepts  # y intercepts of each line segment
```

4 Examples

Simple examples are provided for the following use-cases:

1. fit with explicit breakpoint locations
2. fit for specified number of line segments
3. force the fit through data points
4. use a custom optimization routine to find the optimal breakpoint locations

For additional examples, please look in the examples folder within the source which is available at https://github.com/cjekel/piecewise_linear_fit_py.

To get started with the examples: first import the necessary libraries, then copy the x and y data, and finally initialize the fitting object as *model*.

```
import numpy as np
import pwlf

x = np.array([1., 2., 3., 4., 5., 6., 7., 8., 9., 10., 11.,
              12., 13., 14., 15.])
y = np.array([5., 7., 9., 11., 13., 15., 28.92, 42.81, 56.7,
              70.59, 84.47, 98.36, 112.25, 126.14, 140.03])

model = pwlf.PiecewiseLinFit(x, y)
```

4.1 Fit model with explicit breakpoint locations

The most basic fit is for explicit breakpoint locations by solving the least squares problem presented in Eqn. 7. The example finds the best continuous piecewise linear function that has breakpoint locations at $[0.0, 7.0, 16.0]$. The

fit_with_breaks function is used to perform the least squares fit. The following code performs the fit.

```
breakpoints = [0.0, 7.0, 16.0]
model.fit_with_breaks(breakpoints)
```

Prediction from a fitted model just requires calling the *predict* method on new x locations. The following code evaluates the model on 100 new $\hat{x}$ locations within the domain of x.

```
x_hat = np.linspace(x.min(), x.max(), 100)
y_hat = model.predict(x_hat)
```

The resulting fit and data can be (optionally) plotted using the matplotlib library. The resulting fit is shown in Fig. 1.

```
import matplotlib.pyplot as plt
plt.figure()
plt.plot(x, y, 'o')
plt.plot(x_hat, y_hat, '-')
plt.grid()
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

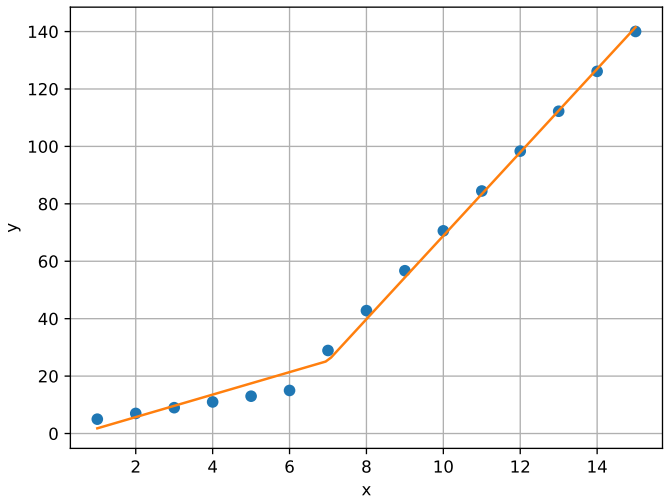

Figure 1: Example of fiting a continuous piecewise linear function with breakpoints occruing at $[0.0, 7.0, 16.0]$.

4.2 Fit for specified number of line segments

The breakpoint locations are unknown in many cases, however with the given example it appears obvious that there are two distinct linear regions. To find the breakpoint locations for two line segments run

```
breakpoints = model.fit(2)
```

where breakpoints is a numpy array containing the optimal breakpoint locations. The result of this fit is shown in Fig. 2, and the resulting breakpoint locations occur at $[1.0, 6.0, 15.0]$.

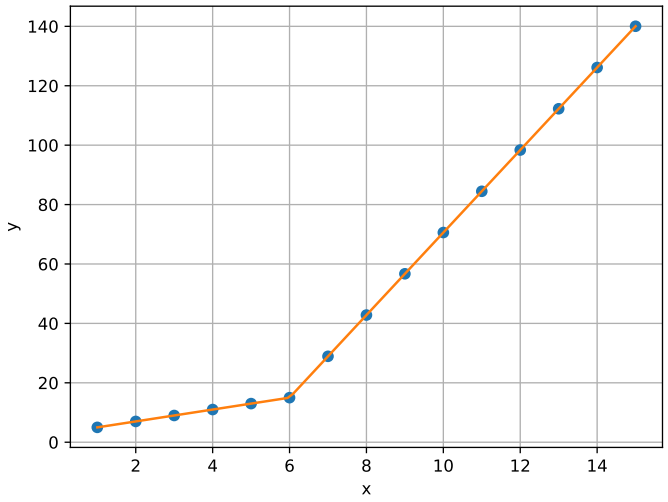

Figure 2: Example of fitting two continuous piecewise line segments to a simple data set.

4.3 Forcing fit through data points

It may sometimes be desirable to force the continuous piecewise linear function through a particular point, or set of points. Such a fit is done by specifying x_c and y_c while performing a fit. The following code finds the best two continuous piecewise lines, such that the model goes through the point $(0.0, 0.0)$. The result is shown in Fig. 3.

```
model.fit(2, x_c=[0.0], y_c=[0.0])
```

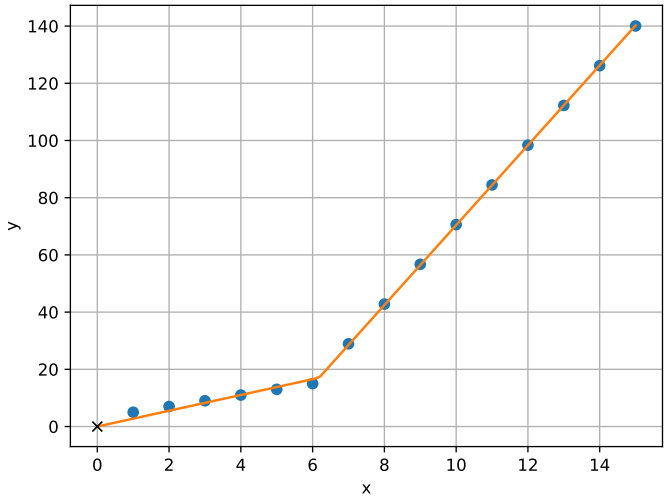

Figure 3: Example of finding the best two continuous piecewise lines that go through the point $(0.0, 0.0)$.

4.4 Using a custom optimization routine

It is possible to find the optimal breakpoint locations using your favorite optimization algorithm. First, run *use_custom_opt* to specify the desired number of line segments. Then, pass *fit_with_breaks_opt* as the objective function to your favorite optimization routine. The following example uses SciPy's minimize to find the best breakpoint locations for two line segments. There is only one variable to optimize, because *pwlf* assumes that the first and last breakpoint occur at the minimum and maximum x data points.

```
from scipy.optimize import minimize
model.use_custom_opt(2)
guess = [5.0]  # guess the breakpoint location
res = minimize(model.fit_with_breaks_opt, guess)
```

5 Conclusion

A methodology was presented for fitting continuous piecewise linear functions to 1D data. If breakpoints (or the termination of each line segment locations) are known, then a simple least squares fit is performed to find the best continuous piecewise linear function. If the breakpoints are unknown but the desired number of line segments is known, then optimization is used to find the breakpoint locations of the best continuous piecewise linear function. This is a double optimization process. The outer loop is attempting to find the best breakpoint locations, while the inner loop is performing a least squares fit to find the best $\boldsymbol{\beta}$ model parameters given some breakpoint locations. This methodology of fitting continuous piecewise linear functions, as well as various

statistics associated with this particular regression model have been discussed in detail. A few examples of the basic usage of *pwlf* was described in this paper. Additionally, there are a number of other examples available online at https://github.com/cjekel/piecewise_linear_fit_py

Acknowledgments

Charles F. Jekel would like to acknowledge University of Florida's Graduate Preeminence Award and U.S. Department of Veterans Affairs' Educational Assistance program for providing funding for his PhD.

Thanks to Raphael Haftka for his numerous comments related to optimization and linear regression.

References

[1] N. M. Fyllas, S. Patiño, T. R. Baker, G. Bielefeld Nardoto, L. A. Martinelli, C. A. Quesada, R. Paiva, M. Schwarz, V. Horna, L. M. Mercado, A. Santos, L. Arroyo, E. M. Jiménez, F. J. Luizão, D. A. Neill, N. Silva, A. Prieto, A. Rudas, M. Silviera, I. C. G. Vieira, G. Lopez-Gonzalez, Y. Malhi, O. L. Phillips, and J. Lloyd, "Basin-wide variations in foliar properties of Amazonian forest: phylogeny, soils and climate," *Biogeosciences*, vol. 6, no. 11, pp. 2677–2708, 2009. [Online]. Available: https://www.biogeosciences.net/6/2677/2009/ 1

[2] C. Ocampo-Martinez and V. Puig, "Piece-wise linear functions-based model predictive control of large-scale sewage systems," pp. 1581–1593, 2010. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/iet-cta.2009.0206 1

[3] R. Heinkelmann, J. Böhm, S. Bolotin, G. Engelhardt, R. Haas, R. Lanotte, D. S. MacMillan, M. Negusini, E. Skurikhina, O. Titov, and H. Schuh, "VLBI-derived troposphere parameters during CONT08," *Journal of Geodesy*, vol. 85, no. 7, pp. 377–393, jul 2011. [Online]. Available: https://doi.org/10.1007/s00190-011-0459-x 1

[4] S. Klikovits, A. Coet, and D. Buchs, "ML4CREST: Machine Learning for CPS Models." 1

[5] G. Villarini, J. A. Smith, and G. A. Vecchi, "Changing Frequency of Heavy Rainfall over the Central United States," *Journal of Climate*, vol. 26, no. 1, pp. 351–357, 2013. [Online]. Available: https://doi.org/10.1175/JCLI-D-12-00043.1 1

[6] J. Ollerton, H. Erenler, M. Edwards, and R. Crockett, "Extinctions of aculeate pollinators in Britain and the role of large-scale agricultural changes," *Science*, vol. 346, no. 6215, pp. 1360–1362, 2014. [Online]. Available: http://science.sciencemag.org/content/346/6215/1360 1

[7] E. Jauk, M. Benedek, B. Dunst, and A. C. Neubauer, "The relationship between intelligence and creativity: New support for the threshold hypothesis by means of empirical breakpoint detection," *Intelligence*, vol. 41, no. 4, pp. 212–221, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016028961300024X 1

[8] V. M. R. Muggeo, "Estimating regression models with unknown break-points," *Statistics in Medicine*, vol. 22, no. 19, pp. 3055–3071, 2003. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.1545 1

[9] N. Golovchenko, "Least-squares fit of a continuous piecewise linear function," 2004. 2

[10] S. Boyd and L. Vandenberghe, *Introduction to Applied Linear Algebra*. Cambridge University Press, 2018. [Online]. Available: https://web.stanford.edu/{~}boyd/vmls/vmls.pdf 5

[11] A. Coppe, R. T. Haftka, and N. H. Kim, "Uncertainty Identification of Damage Growth Parameters Using Nonlinear Regression," *AIAA Journal*, vol. 49, no. 12, pp. 2818–2821, dec 2011. [Online]. Available: http://dx.doi.org/10.2514/1.J051268 6

[12] R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook, *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*, 4th ed., ser. Wiley Series in Probability and Statistics. Wiley, 2016. 6, 7

[13] R. Storn and K. Price, "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, dec 1997. [Online]. Available: https://doi.org/10.1023/A:1008202821328 7

[14] E. Jones, T. Oliphant, P. Peterson, and Others, "SciPy: Open source scientific tools for Python." [Online]. Available: http://www.scipy.org/ 7, 8

[15] V. M. R. Muggeo, "Segmented: an R package to fit regression models with broken-line relationships," *R news*, vol. 8, no. 1, pp. 20–25, 2008. 8

[16] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, 1995. 8

[17] J. F. Schutte, R. T. Haftka, and B. J. Fregly, "Improved global convergence probability using multiple independent optimizations," *International Journal for Numerical Methods in Engineering*, vol. 71, no. 6, pp. 678–702, dec 2006. [Online]. Available: https://doi.org/10.1002/nme.1960 8

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. 9