## CONTENTS:

1  piecewise-regression (aka segmented regression) in python  
2  Installation  
3  Getting started  
4  Example  
5  How It Works  
6  Model Selection  
7  Testing  
8  Requirements  
9  Community Guidelines and Contributing  
10  Installing From Source  
11  Documentation  
   11.1  API  
12  Indices and tables  
  Python Module Index  
  Index
PIECEWISE-REGRESSION (AKA SEGMENTED REGRESSION) IN PYTHON

piecewise-regression  fitting straight line models with breakpoints

Author  Charlie Pilgrim

Version  1.2.1

Github  https://github.com/chasmani/piecewise-regression


Easy-to-use piecewise regression (aka segmented regression) in Python. For fitting straight lines to data where there are one or more changes in gradient (known as breakpoints). Based on Muggeo’s paper “Estimating regression models with unknown break-points” (2003).

When using the package, please cite the accompanying paper.

Example:
Chapter 1. piecewise-regression (aka segmented regression) in python

Code examples below, and more in this Google Colab Jupyter Notebook.
You can install piecewise-regression using python’s pip package index.

```
pip install piecewise-regression
```

The package is tested on Python 3.7, 3.8 and 3.9.
The package requires some x and y data to fit. You need to specify either a) some initial breakpoint guesses as `start_values` or b) how many breakpoints you want to fit as `n_breakpoints` (or both). Here is an elementary example, assuming we already have some data x and y:

```python
import piecewise_regression
pw_fit = piecewise_regression.Fit(x, y, n_breakpoints=2)
pw_fit.summary()
```
For demonstration purposes, substitute with your own data to fit.

1. Start-off generating some data with a breakpoint:

```python
import piecewise_regression
import numpy as np

alpha_1 = -4
alpha_2 = -2
constant = 100
breakpoint_1 = 7
n_points = 200
np.random.seed(0)
xx = np.linspace(0, 20, n_points)
yy = constant + alpha_1*xx + (alpha_2-alpha_1) * np.maximum(xx - breakpoint_1, 0) +
    np.random.normal(size=n_points)
```

2. Fit the model:

```python
# Given some data, fit the model
pw_fit = piecewise_regression.Fit(xx, yy, start_values=[5], n_breakpoints=1)
# Print a summary of the fit
pw_fit.summary()
```

Example output:

### Breakpoint Regression Results

<table>
<thead>
<tr>
<th></th>
<th>200</th>
<th>4</th>
<th>196</th>
<th>193.264</th>
<th>46201.8</th>
<th>0.995817</th>
<th>0.995731</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. Observations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. Model Parameters</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Degrees of Freedom</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Res. Sum of Squares</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Sum of Squares</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R Squared</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adjusted R Squared</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Converged</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Estimate | Std Err | t    | P>|t| | [0.025 | 0.975] |
|----------|---------|------|------|-------|--------|
|         |         |      |      |       |        |

(continues on next page)
These alphas (gradients of segments) are estimated from betas (change in gradient).

Davies test for existence of at least 1 breakpoint: p=5.13032e-295 (e.g. p<0.05 means reject null hypothesis of no breakpoints at 5% significance)

This includes estimates for all the model variables, along with confidence intervals. The Davies test is a hypothesis test for the existence of at least one breakpoint, against the null hypothesis of no breakpoints.

3. Optional: Plotting the data and model results:

```python
import matplotlib.pyplot as plt

# Plot the data, fit, breakpoints and confidence intervals
pw_fit.plot_data(color="grey", s=20)
# Pass in standard matplotlib keywords to control any of the plots
pw_fit.plot_fit(color="red", linewidth=4)
pw_fit.plot_breakpoints()
pw_fit.plot_breakpoint_confidence_intervals()
plt.xlabel("x")
plt.ylabel("y")
plt.show()
plt.close()
```
You can extract data as well:

```python
# Get the key results of the fit
pw_results = pw_fit.get_results()
pw_estimates = pw_results['estimates']
```
The package implements Muggeo’s iterative algorithm (Muggeo “Estimating regression models with unknown breakpoints” (2003)) to find breakpoints quickly. This method simultaneously fits breakpoint positions and the linear models for the different fit segments, and it gives confidence intervals for all the model estimates. See the accompanying paper for more details.

Muggeo’s method doesn’t always converge on the best solution - sometimes, it finds a locally optimal solution or doesn’t converge at all. For this reason, the Fit method also implements a process called bootstrap restarting which involves taking a bootstrapped resample of the data to try to find a better solution. The number of times this process runs can be controlled with n_boot. To run the Fit without bootstrap restarting, set n_boot=0.

If you do not have (or do not want to use) initial guesses for the number of breakpoints, you can set it to n_breakpoints=3, and the algorithm will randomly generate start_values. With a 50% chance, the bootstrap restarting algorithm will either use the best currently converged breakpoints or randomly generate new start_values, escaping the local optima in two ways in order to find better global optima.

As is often the case with fitting non-linear models, even with these measures, the algorithm is not guaranteed to converge to a global optimum. However, increasing n_boot raises the probability of global convergence at the cost of computation time.
In addition to the main Fit tool, the package also offers a ModelSelection option based on the Bayesian Information Criterion (BIC). This additional tool is experimental and not as thorough as the main Fit function. In particular, the models are generated with random start_values, which can influence the model fit and give different values for the BIC. The tool can help explore other possible models but should not be used to choose the best model at this time.

```python
ms = piecewise_regression.ModelSelection(x, y, max_breakpoints=6)
```

Example output:

<table>
<thead>
<tr>
<th>n_breakpoints</th>
<th>BIC</th>
<th>converged</th>
<th>RSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>421.09</td>
<td>True</td>
<td>1557.4</td>
</tr>
<tr>
<td>1</td>
<td>14.342</td>
<td>True</td>
<td>193.26</td>
</tr>
<tr>
<td>2</td>
<td>22.825</td>
<td>True</td>
<td>191.23</td>
</tr>
<tr>
<td>3</td>
<td>24.169</td>
<td>True</td>
<td>182.59</td>
</tr>
<tr>
<td>4</td>
<td>29.374</td>
<td>True</td>
<td>177.73</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>False</td>
<td></td>
</tr>
</tbody>
</table>

Minimum BIC (Bayesian Information Criterion) suggests the best model

The data of the model fits can be accessed in

```python
ms.models
```

For a robust comparison, you could run the ModelSelection tools many times and take the lowest BIC for each model.
The package includes comprehensive tests.

To run all tests, from the main directory run (requires the pytest library):

```bash
pytest
```

To get code coverage, run (requires pytest and pytest-cov libraries):

```bash
pytest --cov=./
```

There are also a series of simulation tests that check the estimates have realistic confidence intervals, and the Davies test gives realistic p-values. These can be found in the folder "tests-manual".
See requirements.txt for specific version numbers. Required packages, and their uses are:

- matplotlib for plotting.
- numpy for simple data handling and data transformations.
- scipy for statistical tests including using t-distributions and Gaussians.
- statsmodels for performing ordinary least squares.
COMMUNITY GUIDELINES AND CONTRIBUTING

We welcome community participation!

Sourced from Open Source Guide: How to contribute.

Open an issue in the following situations:

• Report an error you can’t solve yourself
• Discuss a high-level topic or idea (for example, community, vision or policies)
• Propose a new feature or other project ideas

Tips for communicating on issues:

• If you see an open issue that you want to tackle, comment on the issue to let people know you’re on it. That way, people are less likely to duplicate your work.
• If an issue was opened a while ago, it’s possible that it’s being addressed somewhere else, or has already been resolved, so comment to ask for confirmation before starting work.
• If you opened an issue, but figured out the answer later on your own, comment on the issue to let people know, then close the issue. Even documenting that outcome is a contribution to the project.

Open a pull request in the following situations:

• Submit trivial fixes (for example, a typo, a broken link or an obvious error)
• Start work on a contribution that was already asked for, or that you’ve already discussed, in an issue

Tips for submitting PRs:

• Reference any relevant issues or supporting documentation in your PR (for example, “Closes #37.”)
• Include screenshots of the before and after if your changes include differences in HTML/CSS. Drag and drop the images into the body of your pull request.
• Test your changes by running them against any existing tests if they exist and create new ones when needed. Whether tests exist or not, make sure your changes don’t break the existing project.
• Contribute in the style of the project to the best of your abilities.
To install from source:

```
git clone https://github.com/chasmani/piecewise-regression
cd piecewise_regression
python3 setup.py install --user
```
Full docs, including an API reference.

### 11.1 API

#### 11.1.1 Main

The main module includes the `Fit` function, which runs the bootstrap restarting algorithm.

```python
class piecewise_regression.main.Fit(xx, yy, start_values=None, n.breakpoints=None, n.boot=100,
   verbose=False, max.iterations=30, tolerance=1e-05,
   min.distance.between.breakpoints=0.01,
   min.distance.to.edge=0.02)
```

Fit a piecewise (segmented) regression model to data. Uses bootstrap restarting to avoid local minima. Requires either `n.breakpoints` or `start.values`. If no `start_values` are given, they are instead uniformly randomly generated across range of data. This is the main user facing object and input data is validated mainly at this level.

**Parameters**

- **xx (list of floats)** – Data series in x-axis for fitting (same axis as the breakpoints)
- **yy (list of floats)** – Data series in y-axis for fitting.
- **n.breakpoints (positive int)** – The number of breakpoints to fit.
- **start.values (list floats)** – A list of initial guesses for the breakpoints.
- **n.boot (non-negative int)** – How many times to run the bootstrap restarting procedure. Set to zero for no bootstrap restarting.
- **verbose (bool)** – If True, prints out updates to the terminal.
- **max.iterations (positive int)** – How many iterations before stopping if not converged, in the Muggeo iterative procedure.
- **tolerance (positive float)** – How close breakpoints from previous iterations must be to consider converged.
- **min.distance.between.breakpoints (positive float)** – The minimum allowed distance between breakpoints, as a proportion of the data range.
- **min.distance.to.edge** – The minimum allowed distance from the edge of data to a breakpoint, as a proportion of the data range.

```python
bootstrap_data(xx, yy)
```

Non parametric bootstrap, randomly sample data points with replacement. Return bootstrapped data of same length as original data.
Parameters

- **xx** *(list of floats)* – Data series in x-axis.
- **yy** *(list of floats)* – Data series in y-axis.

*bootstrap_restarting()*

The main fitting algorithm. Begins by doing a fit based on Muggeo’s algorithm. if n_boot = 0 we stop there. Otherwise we do some bootstrap restarting. Bootstrap Restarting escapes local minima. Each bootstrap restart:

- We take the best current breakpoints, and get new data by running a non-parametric bootstrap by resampling data.

  - Then run a Muggeo fit on the new data and best current breakpoints. This gives new breakpoint values.
  - Then run a Muggeo fit again with the original data and these new breakpoint values.
  - Throughout, keep track of the history of fits and the best_muggeo fit that converged - defined as the lowest residual sum of squares.

*get_results()*

Return a small dictionary with key results form the fit. Useful for using this code in a larger analysis. E.g. ModelSelection

*plot()*

Plot the full fit including the data, fitted model, breakpoint positions and breakpoint confidence intervals. Doesn’t allow control of matplotlib kwargs for style changes.

*plot_best_muggeo_breakpoint_history(**kwargs)**

Plot the history of the breakpoints as they iterate. History of the best_muggeo fit.

*plot_bootstrap_restarting_history(**kwargs)**

Plot the history of the breakpoint values as they iterate. History of the bootstrap restarting procedure.

*plot_bootstrap_restarting_rss_history(**kwargs)**

Plot the history of the residual sum of squares. History of the bootstrap restarting algorithm.

*plot_breakpoint_confidence_intervals(**kwargs)**

Plot the breakpoint confidence intervals as vertical shaded regions. Passes kwargs to the matplotlib function, e.g. color="red".

*plot_breakpoints(**kwargs)**

Plot the breakpoint locations as vertical lines. Passes kwargs to the matplotlib function, e.g. color="red".

*plot_data(**kwargs)**

Plot the data as a scatter plot. Passes any kwargs to the matplotlib scatter function, e.g. color="red".

*plot_fit(**kwargs)**

Plot the fitted model as a series of straight lines. Passes any kwargs to the matplotlib plot function, e.g. color="red".

*summary()*

Print a summary of the best fit, along the lines of the summary given by python’s statsmodels OLS fit.

*class piecewise_regression.main.Muggeo(xx, yy, n_breakpoints, start_values=None, verbose=False, max_iterations=30, tolerance=1e-05, min_distance_between_breakpoints=0.01, min_distance_to_edge=0.02)*

Muggeo’s iterative segmented regression method. This is a simple version. Errors are handled at a higher level in the Fit object. See Muggeo (2003). If the breakpoints get too close together, or get outside (or near the edge)
of the data range, the algorithm is stopped because this is likely to be a local minimum that is difficult to escape, as well as possibly generating errors in the iterative procedure.

**Parameters**

- `xx (list of floats)` – Data series in x-axis for fitting (same axis as the breakpoints)
- `yy (list of floats)` – Data series in y-axis for fitting
- `n_breakpoints (positive int)` – The number of breakpoints to fit
- `start_values (list floats)` – A list of initial guesses for the breakpoints
- `verbose (bool)` – If True, prints out updates to the terminal
- `max_iterations (positive int)` – How many iterations before stopping if not converged
- `tolerance (positive float)` – How close breakpoints from previous iterations must be to consider converged.
- `min_distance_between_breakpoints (positive float)` – The minimum allowed distance between breakpoints, as a proportion of the data range.
- `min_distance_between_breakpoints` – The minimum allowed distance from the edge of data to a breakpoint, as a proportion of the data range.

**fit()**
Runs the breakpoint iterative procedure

**stop_or_not()**
Test to see if the iterative procedure should stop. Stop if it’s converged, if max_iterations reached, or the breakpoints fall into values that are too close together or outside of the allowed range.

**class piecewise_regression.main.NextBreakpoints(xx, yy, current_breakpoints)**
One iteration of Muggeo’s segmented regression algorithm. Gets the next breakpoints. Also calculates interesting statistics. This expects data validation and error handling are done at a higher level.

**Parameters**

- `xx (list)` – Data series in x-axis for fitting (same axis as the breakpoints)
- `yy (list)` – Data series in y-axis for fitting
- `current_breakpoints (list)` – The starting breakpoints for this iteration

**breakpoint_fit()**
Fit the linear approximation given the current breakpoint guesses. Sets the next breakpoints and the params from the fit. The params are of the form [c, a, beta_hats, gamma_hats]

**calculate_all_confidence_intervals()**
Calculate all confidence intervals, based on t-distribution and standard errors.

**calculate_all_estimates()**
Extract estimates from the params and saves in self.estimates

**calculate_all_standard_errors()**
Calculate standard errors for all the variables of interest. Save to the self.estimates dictionary

**calculate_all_t_stats()**
Get t stats for all the estimators

**calculate_bayesian_information_criterion()**
Calculates the Bayesian Information Criterion of the fitted model. Assuming normal noise, uses the standard version for OLS models. This should hold for breakpoint regression models, because the BIC is based on the likelihood of the data given the model. That likelihood function doesn’t involve the breakpoint values
- it just depends on distances of the data to the fitted model predictions. Also depends on the error in the noise term being constant.

```python
calculate_r_squared()
Calculate R squared from the fitted model.
```

```python
get_alpha_standard_errors()
Get the standard errors for the alphas (gradients of segments)
```

```python
get_beta_standard_errors()
Get the beta estimates standard errors from the covariance matrix
```

```python
get_bp_standard_errors()
Get the standard errors of the breakpoints. Considering \( bp = \gamma/beta + bp_0 \), the standard error of the breakpoint estimates can be found using the ratio/delta method. See e.g. Muggeo (2003) for clarification
```

```python
get_const_standard_error()
Get the constant standard error from the covariance matrix
```

```python
get_predicted_yy()
Get the model predictions for each of the xx data points
```

### 11.1.2 Model selection

The model selection module is experimental. It compares models with different \( n\_breakpoints \) using the Bayesian Information Criterion.

```python
class piecewise_regression.model_selection.ModelSelection(xx, yy, max_breakpoints=10,
             n_boot=100, max_iterations=30,
             tolerance=1e-05,
             min_distance_between_breakpoints=0.01,
             min_distance_to_edge=0.02,
             verbose=True)
```

Experimental - uses simple BIC based on simple linear model.

### 11.1.3 Davies test

Implements the Davies hypothesis test for existence of at least one breakpoint.

```python
piecewise_regression.davies.davies_test(xx, yy, k=10, alternative='two_sided')
```

Significance test for the existence of a breakpoint Null hypothesis is that there is no breakpoint, or that the change in gradient is zero. Alternative hypothesis is that there is a breakpoint, with a non-zero change in gradient. The change is gradient is a function of the breakpoint position. The breakpoint position is a nuisance parameter that only exists in the alternative hypothesis. Based on Davies (1987), “Hypothesis Testing when a nuisance parameter is present only under the alternative”.

**Parameters**

- **xx** (list of floats) – Data series in x-axis (same axis as the breakpoints).
- **yy** (list of floats) – Data series in y-axis.
- **k** (int) – A control parameter that determines the number of points to consider within the xx range.
- **alternative** (str. One of "two_sided", "less", "greater") – Whether to consider a two-sided hypothesis test, or a one sided test with change of gradient greater or less than zero. For existence of a breakpoint, use “two-sided”.
Compute a test statistic for the Davies test for the p-value of existence of a breakpoint. Based on Davies(1987) “Hypothesis Testing when a nuisance parameter is present only under the alternative”. All the variables in this function are as named and described in that paper.

**Parameters**

- **xx_davies** *(list of floats)* – Data series in x-axis (same axis as the breakpoints).
- **yy_davies** *(list of floats)* – Data series in y-axis.
- **theta** *(float)* – A test value from within the range of data in xx.
INDICES AND TABLES

- genindex
- modindex
- search
p

piecewise_regression.davies, 26
piecewise_regression.main, 23
piecewise_regression.model_selection, 26
INDEX

B
bootstrap_data() (piecewise_regression.main.Fit method), 23
bootstrap_restarting() (piecewise_regression.main.Fit method), 24
breakpoint_fit() (piecewise_regression.main.NextBreakpoints method), 25

calculate_all_confidence_intervals() (piecewise_regression.main.NextBreakpoints method), 25
calculate_all_estimates() (piecewise_regression.main.NextBreakpoints method), 25
calculate_all_standard_errors() (piecewise_regression.main.NextBreakpoints method), 25
calculate_all_t_stats() (piecewise_regression.main.NextBreakpoints method), 25
calculate_bayesian_information_criterion() (piecewise_regression.main.NextBreakpoints method), 25
calculate_r_squared() (piecewise_regression.main.NextBreakpoints method), 26

davies_test() (in module piecewise_regression.davies), 26

F
Fit (class in piecewise_regression.main), 23
fit() (piecewise_regression.main.Muggeo method), 25

G
get_alpha_standard_errors() (piecewise_regression.main.NextBreakpoints method), 26
get_beta_standard_errors() (piecewise_regression.main.NextBreakpoints method), 26
get_bp_standard_errors() (piecewise_regression.main.NextBreakpoints method), 26
get_const_standard_error() (piecewise_regression.main.NextBreakpoints method), 26
get_predicted_yy() (piecewise_regression.main.NextBreakpoints method), 26
get_results() (piecewise_regression.main.Fit method), 24
get_test_statistic() (in module piecewise_regression.davies), 26

M
ModelSelection (class in piecewise_regression.model_selection), 26

N
NextBreakpoints (class in piecewise_regression.main), 25

P
piecewise_regression.davies module, 26
piecewise_regression.main module, 23
piecewise_regression.model_selection module, 26
plot() (piecewise_regression.main.Fit method), 24
plot_best_muggeo_breakpoint_history() (piecewise_regression.main.Fit method), 24
plot_bootstrap_restarting_history() (piecewise_regression.main.Fit method), 24
plot_bootstrap_restarting_rss_history()
   (piecewise_regression.main.Fit method), 24
plot_breakpoint_confidence_intervals()
   (piecewise_regression.main.Fit method), 24
plot_breakpoints()
   (piecewise_regression.main.Fit method), 24
plot_data()
   (piecewise_regression.main.Fit method), 24
plot_fit()
   (piecewise_regression.main.Fit method), 24

S
stop_or_not()
   (piecewise_regression.main.Muggeo method), 25
summary()
   (piecewise_regression.main.Fit method), 24