Python Piecewise Regression Package

The piecewise regression package fits a linear regression function to a dataset that has instantaneous rates of change (breakpoints)

How it works

The algorithm was first developed by Vito M. R. Muggeo and explained further in his paper "Estimating regression models with unknown break-points." The model with only one breakpoint looks like this:

$$y = \alpha x + c + \beta(x - \psi)H(x - \psi) + \zeta$$

Where x and y is the dataset, α and c are the slope and intercept of the first segment, β is the change in slope to the second segment and ψ is the breakpoint position. H is something called a Heaviside step function which basically denotes when the function has switched from the first segment to the second. ζ is the noise constant.

This isn't a linear function so they use a Taylor expansion around an initial guessed breakpoint location and it looks something like this:

$$y \approx \alpha x + c + \beta(x - \psi_0)H(x - \psi_0) - \beta(\psi - \psi_0)H(x - \psi_0) + \zeta$$

They then iterate chosing a new breakpoint location at each iteration and find the linear regression of each segment by statsmodels library.

How it's used

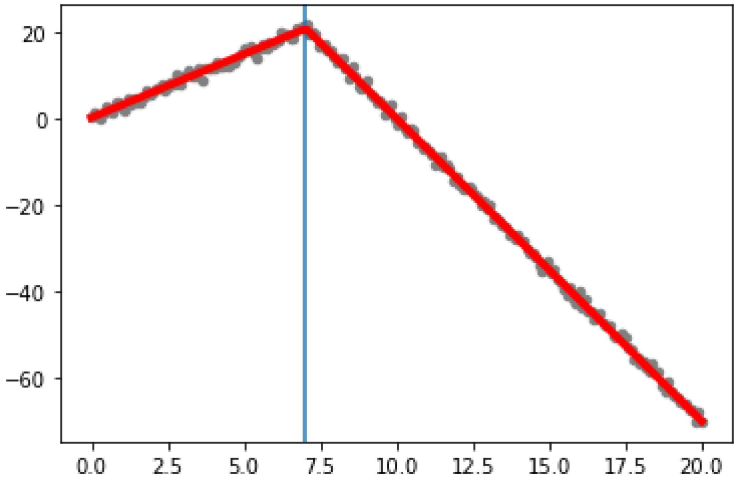
The paper mentions a couple places where it could be used such as medical interventions, ecological thresholds and geological phase transitions. I thought maybe it could be used in the stock market to analyze trends. However, this package can be useful for any situation where one might need to analyze data where there are immediate changes in slope.

In [73]:
```
%%capture
import sys
!{sys.executable} -m pip install piecewise_regression
import piecewise_regression
import numpy as np
import matplotlib.pyplot as plt
```

Example

In [75]:
```
#Creates random dataset along two linear functions
alpha_1 = 3
alpha_2 = -7
n_points = 200
x = np.linspace(0, 20, n_points)
y = alpha_1*x + (alpha_2-alpha_1) * np.maximum(x - breakpoint_1, 0) + np.rand

#Uses the piecewise regression package to fit and plot the data
pw_fit = piecewise_regression.Fit(x, y, n_breakpoints=1)
pw_fit.plot_data(color="grey", s=20)
pw_fit.plot_breakpoints()
pw_fit.plot_fit(color="red", linewidth=4)

results = pw_fit.get_results()
print("Slope of first function: %f" % results['estimates']['alpha1']['estimat
print("Slope of second function: %f" % results['estimates']['alpha2']['estima
```

```
Slope of first function: 2.945801
Slope of second function: -7.023650
```



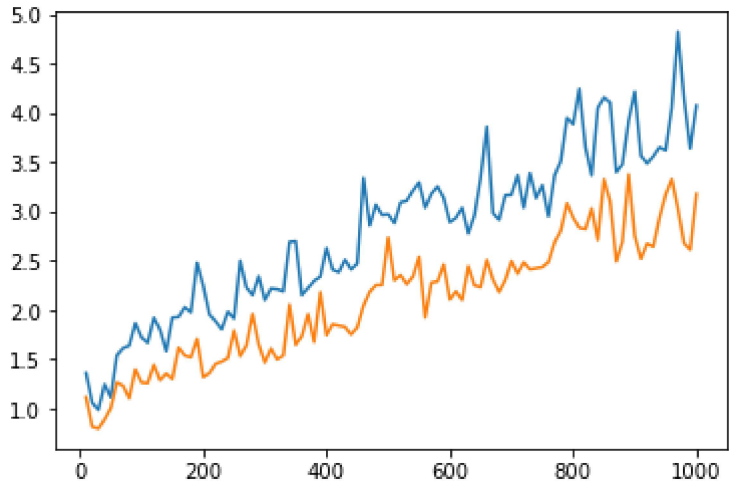
It's important to note that the package requires either a guess of the location of the breakpoints (instantanous rates of change) or number of breakpoints. I'm also fairly certain that providing only the number of breakpoints forces the algorithm to spend more time search for it. You'll also notice the package has its own set of plot functions, which are very helpful.

Speed and Slope Error Tests

I'm going to conduct two test, one without a breakpoint guess and one with just the number of breakpoints. My hypothesis is that the duration of the fit function takes longer without an accurate breakpoint guess. I'm using the same type of piecewise function as before, but with different numbers of points per test. As for the error, I'm going to measure how the number of points affects the accuracy of the estimate of the slope.

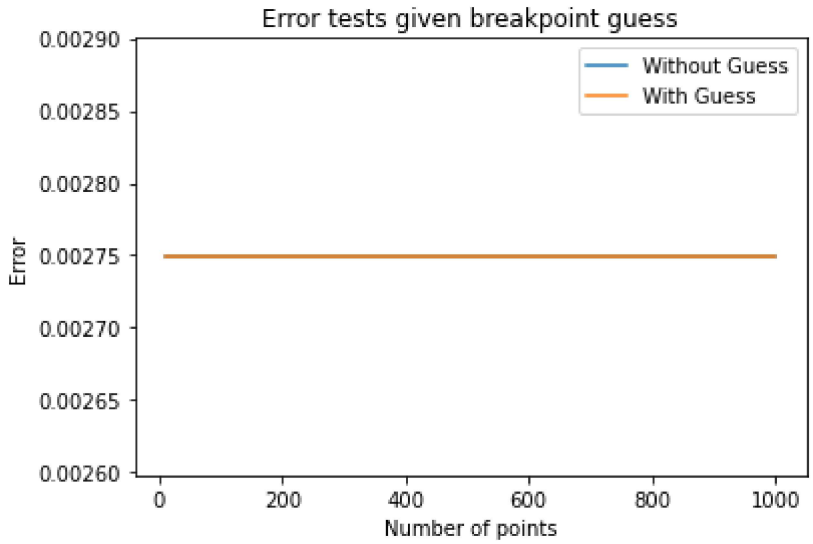
In [96]:

```
import time

numTests = 100
tests = np.array(range(1, numTests+1))*10

durationsA = []
errorA = []
durationsB = []
errorB = []

breakpointEst = 7

for i in tests:

    alpha_1 = 3
    alpha_2 = -7
    n_points = i
    x = np.linspace(0, 20, n_points)
    yA = alpha_1*x + (alpha_2-alpha_1) * np.maximum(x - breakpoint_1, 0) + np
    yB = alpha_1*x + (alpha_2-alpha_1) * np.maximum(x - breakpoint_1, 0) + np

    t0A = time.time()
    pw_fitA = piecewise_regression.Fit(x, yA, n_breakpoints=1)
    t1A = time.time()

    t0B = time.time()
    pw_fitB = piecewise_regression.Fit(x, yB, start_values=[breakpointEst])
    t1B = time.time()

    resultsA = pw_fitA.get_results()
    ea1A = results['estimates']['alpha1']['estimate']
    ea2A = results['estimates']['alpha2']['estimate']
    resultsB = pw_fitB.get_results()
    ea1B = results['estimates']['alpha1']['estimate']
    ea2B = results['estimates']['alpha2']['estimate']

    errorA.append(np.mean([abs(alpha_1 - ea1A), abs(alpha_2 - ea2A)]))
    durationsA.append(t1A-t0A)
    errorB.append(np.mean([abs(alpha_1 - ea1B), abs(alpha_2 - ea2B)]))
    durationsB.append(t1B-t0B)

plt1 = plt.plot(tests, durationsA, label="Without Guess")
plt2 = plt.plot(tests, durationsB, label ="With Guess")
```



In [97]:
```
plt1 = plt.plot(tests, errorA, label="Without Guess", )
plt2 = plt.plot(tests, errorB, label ="With Guess")
plt.xlabel("Number of points")
plt.ylabel("Error")
plt.title("Error tests given breakpoint guess")
plt.legend()
```

Out[97]: <matplotlib.legend.Legend at 0x284ff10b070>



It's apparent from this figure that an accurate guess benefits the speed at which the fit function fits the linear regression. This means the algorithm can be optimized by inputing guessed breakpoint values. With that being said, they both seem to run at O(n) time. Of course larger datasets will tend to be slower compuationaly. The error seems very consistant with and without breakpoint guesses. The error is also constant for at least 10 points of data. However, this might be due to the dataset.

Questions

Like I mentioned before, I'm curious how this package could be applied to stock market analysis or other fields. I imagine if you feed the piecewise regression function a set of price points and varried the number of breakpoints the algorithm uses, you could incrementaly analyze different regression models for the same data.

For a group project, I propose testing this package with different sets of data to see how it fits in different ways. The algorithm tends to diverge especially when fed too many breakpoints, so it would be interesting to see to what extent the algorithm fails.

In [ ]: