Multiple Partition Regression Analysis

Rhys Hawkins and Malcolm Sambridge

October 4, 2012

This tutorial describes how to analyse discontinuous data using Monte-Carlo Markov-Chain based regression.

Pre-requisites

This tutorial is for the Python version of the rjmcmc library. The examples rely on the Matplotlib library for plotting. The versions used in the development of this tutorial are as follows:

- Python 2.7.1

- rjmcmc 0.1.0

- Matplotlib 1.1.0

The Data

Similar to the single partition tutorial we will use a non-trivial synthetic dataset with added noise. The difference this time is that we will add a series of discontinuities or step functions and a sign change.

The base function that is used is an exponentially increasing sine wave over the domain 0 . . . 10, ie:

$$y = \text{stepsign}(x) \times e^{\frac{x}{3}} \sin \frac{2x}{3} + \text{step}(x) \qquad (1)$$

Where the step and stepsign functions are defined as follows:

$$\text{step}(x) = \begin{cases} 15 & x < 2.5 \\ -20 & 2.5 \geq x < 5 \\ 0 & otherwise \end{cases} \qquad (2)$$

$$\text{stepsign}(x) = \begin{cases} -1 & x < 2.5 \\ 1 & 2.5 \geq x < 5 \\ -1 & otherwise \end{cases} \qquad (3)$$

The actual dataset unevenly (though with fairly good coverage) samples this function and adds some Gaussian noise and these values are save to an ASCII text file. A plot of the synthetic data points with the true function is shown in Figure 1.
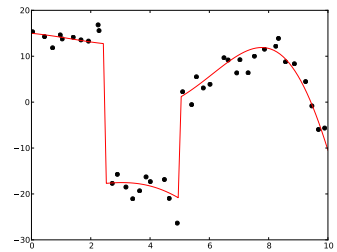

Figure 1: The Synthetic Data

Loading the Data

For those unfamiliar with Python, we discuss briefly the loading of data from a simple ASCII text file. The code for loading the data and doing a simple plot is shown in Listing 1.

Listing 1: Loading the Data

```python
#
# Import the libraries we will need for analysis and plotting.
#
import rjmcmc
import matplotlib
import matplotlib.pyplot

#
# Open our data file which consists of one (x, y) coordinater per line
# separated by whitespace
#
f = open('data.txt', 'r')
lines = f.readlines()

x = []
y = []

for line in lines:
    columns = line.split()

    x.append(float(columns[0]))
    y.append(float(columns[1]))

f.close()

fig = matplotlib.pyplot.figure()

matplotlib.pyplot.plot(x, y, 'ko')

fig.savefig('ch1-loading.pdf', format='PDF')
matplotlib.pyplot.show()
```

The ASCII file contains an x, y pair per line separated by a space. On line 12, we use the built in function open to open the file. One lines 15 and 16 we initialize 2 list that will contain the x and y coordinates in the file. On line 18 we loop through the lines in the file and add each x, y pair to the 2 separate lists.

From line 26 onwards, we plot the data using the matplotlib library and save the plot to a PDF file. The plot resulting from this script can be seen in Figure 2.
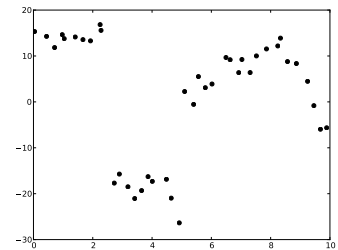

Figure 2: The Plot of the Data

Running the default analysis

For performing a regression analysis on a continuous dataset the function is called regression_part1d. The parameters for this function are mostly the same as the regression_single1d function with the only additional required parameter being pd.

The complete list of parameters for this function are as follows with default values shown where applicable:

dataset The dataset object to run the analysis on. This is an rjmcmc.dataset1d object which wraps the x and y vectors you load from the file and includes individual point noise values. This is the only parameter which doesn't have a default value.

pd The standard deviation for the perturbation of partition boundaries.

burnin = 10000 The number of initial samples to throw away.

total = 50000 The total number of samples to use for the analysis.

max_order = 1 The maximum order of polynomial to use to fit the
data.

xsamples = 100 The number of points to sample along the x direction
for the curve.

ysamples = 100 The number of points to sample along the y directory
for the statistics such as mode, median and confidence intervals.
This is the number of bins for the histograms in the y direction.

confidence_interval = 0.95 The confidence interval to use for mini-
mum and maximum confidence intervals. This should be a value
between 0 and 1.

For this analysis we are only going to use the default values and
the listing is shown in Listing 2.

Listing 2: Running the Default Analysis

```
1   #
2   # Import the libraries we will need for analysis and plotting.
3   #
4   import rjmcmc
5   import matplotlib
6   import matplotlib.pyplot
7
8   #
9   # Open our data file which consists of one (x, y) coordinate per line
10  # separated by whitespace
11  #
12  f = open('data.txt', 'r')
13  lines = f.readlines()
14
15  x = []
16  y = []
17
18  for line in lines:
19      columns = line.split()
20
21      x.append(float(columns[0]))
22      y.append(float(columns[1]))
23
24  f.close()
25
26  #
27  # Set our x range
28  #
29  xmin = 0.0
30  xmax = 10.0
31
32  #
33  # Estimate our error standard deviation
34  #
35  sigma = 3.0
36  n = [sigma] * len(x)
37
38  #
39  # Create the rjmcmc dataset
40  #
41  data = rjmcmc.dataset1d(x, y, n)
42
43  #
44  # Specify the standard deviation for the move partition
45  #
46  pd = 1.0
47
48  #
49  # Run an analysis with reduced max order to only allow linear
50  # segments
51  #
52  burnin = 10000
53  total = 50000
54  max_partitions = 10
55  max_order = 1
56
57  results = rjmcmc.regression_part1d(data,
```

```
58                                     pd ,
59                                     burnin ,
60                                     total ,
61                                     max_partitions ,
62                                     max_order )
63
64   #
65   # Retrieve the mean curve for plotting
66   #
67   xc = results.x()
68   meancurve = results.mean()
69
70   #
71   # Retrieve the partition location and count information
72   #
73   partlocation = results.partition_location_histogram()
74   partcount = results.partitions()
75
76   #
77   # Plot the data with black crosses and the mean with a red line
78   #
79   fig = matplotlib.pyplot.figure(1)
80
81   a = matplotlib.pyplot.subplot(211)
82
83   a.plot(x, y, 'ko', xc, meancurve, 'r-')
84   a.set_xlim(xmin, xmax)
85
86   b = matplotlib.pyplot.subplot(212)
87   b.bar(xc, partlocation, xc[1] - xc[0])
88   b.set_xlim(xmin, xmax)
89
90   fig.savefig('ch2-analyse.pdf', format='PDF')
91
92   fig = matplotlib.pyplot.figure(2)
93
94   a = matplotlib.pyplot.subplot(111)
95   a.hist(partcount, bins=5, range=(0,5), align='left')
96
97   fig.savefig('ch2-analyse-partcount.pdf', format='PDF')
98
99   matplotlib.pyplot.show()
```

The preamble (lines 1 ... 24) consists of loading the file as in the previous section.

An important part of the analysis is estimating the error in the data. This is specified as a error value per data point and can be thought of a weighting as to how well the fit will attempt to fit an individual point. If the value is low, then the fit will be tight and conversely if the value is high then the fit will be loose. On lines 35 and 36 we set a value of 3.0 for all data points. Use this value for now, but try other values greater than 0.0 to see the effect.

On line 41 we construct the dataset1d object from the x, y and n lists we created. These lists must be the same length.

On line 46 we set the value for the pd parameter. With partition modelling, a variable number of discontinuities are trialed at random locations along the x-axis. As part of the fitting process, the locations of these discontinuities are perturbed by an amount determined by sampling from a normal distribution with a standard deviation given by the pd parameter. We will discuss how to choose this value latter, but for now a useful rule of thumb would be to set it to 5 to 10 percent of the range of the data. So in this case our range is 10, so a pd of 1 would seem a good first approximation.

On line 57 we run the analysis with this dataset1d object. The regression_single1d function returns a resultset1d object which contains various results and diagnostics about the analysis. For this simple analysis we simply take the x sampling coordinates and the mean of the fits. And plot the mean with the original data points to

see how representative the mean is. This plot is shown in Figure 3.

In the plot, rather than just plotting the mean of the fits, we have also plotted the histogram of the location of the partition boundaries which indicates the most likely location of the discontinuities. As can be seen in the figure, the histogram highlights the artificial discontinuities created at x = 2.5 and 5.

We have also plotted the histogram of the number of partitions as shown in Figure 4. From this we can infer the likelihood of the number of the partitions in the data.

It should be noted here that there are two discontinuities in function and one less prominent change in gradient in the real data. Looking at the plot of the number of partitions, we see strong support for 4 partitions, as the algorithm has identified the four segments in the original curve.

Increasing the maximum order of the polynomial

Partition modelling and allowing higher order polynomials are in conflict to some degree. A higher order polynomial can fit a discontinuity to some degree and therefore obviate the need to create an extra partition.

To show this, we can increase the maximum order allowed for the polynomials in each partition to 5. This means that the fitting procedure may use up to a 5th order polynomial instead of line segments. The script to do this is shown in Listing 3.

Figure 3: The Default Analysis Plot

Figure 4: The Partition Count Histogram

Listing 3: Increasing the maximum order of the polynomials

```
#
# Import the libraries we will need for analysis and plotting.
#
import rjmcmc
import matplotlib
import matplotlib.pyplot

#
# Open our data file which consists of one (x, y) coordinate per line
# separated by whitespace
#
f = open('data.txt', 'r')
lines = f.readlines()

x = []
y = []

for line in lines:
    columns = line.split()

    x.append(float(columns[0]))
    y.append(float(columns[1]))

f.close()

#
# Set our x range
#
xmin = 0.0
xmax = 10.0

#
# Estimate our error standard deviation
#
sigma = 3.0
n = [sigma] * len(x)

#
# Create the rjmcmc dataset
#
```
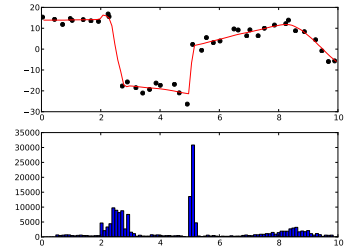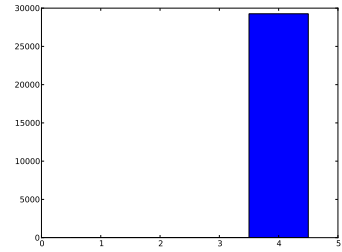
```
41  data = rjmcmc.dataset1d(x, y, n)
42
43  #
44  # Specify the standard deviation for the move partition
45  #
46  pd = 1.0
47
48  #
49  # Run the default analysis
50  #
51  results = rjmcmc.regression_part1d(data, pd)
52
53  #
54  # Retrieve the mean curve for plotting
55  #
56  xc = results.x()
57  meancurve = results.mean()
58
59  #
60  # Retrieve the partition location and count information
61  #
62  partlocation = results.partition_location_histogram()
63  partcount = results.partitions()
64
65  #
66  # Plot the data with black crosses and the mean with a red line
67  #
68  fig = matplotlib.pyplot.figure(1)
69
70  a = matplotlib.pyplot.subplot(211)
71
72  a.plot(x, y, 'ko', xc, meancurve, 'r-')
73  a.set_xlim(xmin, xmax)
74
75  b = matplotlib.pyplot.subplot(212)
76  b.bar(xc, partlocation, xc[1] - xc[0])
77  b.set_xlim(xmin, xmax)
78
79  fig.savefig('ch3-order.pdf', format='PDF')
80
81  fig = matplotlib.pyplot.figure(2)
82
83  a = matplotlib.pyplot.subplot(111)
84  a.hist(partcount, bins=5, range=(0,5), align='left')
85
86  fig.savefig('ch3-orderpartcount.pdf', format='PDF')
87
88  matplotlib.pyplot.show()
```

The results of running this script can be seen in Figures 5 and 6. As can be seen, the fit is more smooth than the previous analysis and supports a fewer number of partitions. In particular we can see that there is some support for no discontinuities, but it is more likely that there are 1 or 2 which matches our artificially created boundaries (the discontinuity at x = 5 is not that large so it can be accommodated by a higher order polynomial). Try changing the maximum allowed order to, say, 2, i.e. allow up to a quadratic, and examine what happens to the Partition Count Histogram.

Confidence

So far we have only plotted the mean of the fits, however this gives only a glimpse as to how the data is being fitted. One way to look at the fitting is to takes samples of the fitting curves. The script to do this is showing in Listing 4.

Listing 4: Sampling the Fitting

```
1   #
2   # Import the libraries we will need for analysis and plotting.
3   #
4   import rjmcmc
5   import matplotlib
6   import matplotlib.pyplot
7   from mpl_toolkits.mplot3d import axes3d, Axes3D
8
9   #
10  # Open our data file which consists of one (x, y) coordinater per line
```
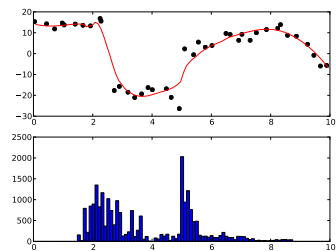

Figure 5: The Order Analysis Plot
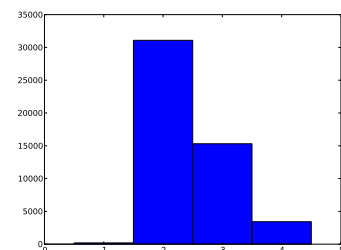

Figure 6: The Partition Count Histogram

```
11   # separated by whitespace
12   #
13   f = open('data.txt', 'r')
14   lines = f.readlines()
15
16   x = []
17   y = []
18
19   for line in lines:
20       columns = line.split()
21
22       x.append(float(columns[0]))
23       y.append(float(columns[1]))
24
25   f.close()
26
27   #
28   # Estimate our error standard deviation
29   #
30   sigma = 3.0
31   n = [sigma] * len(x)
32
33   #
34   # Create the rjmcmc dataset
35   #
36   data = rjmcmc.dataset1d(x, y, n)
37
38   #
39   # Set our x range
40   #
41   xmin = 0.0
42   xmax = 10.0
43
44   #
45   # This is our callback function which samples the curves generated
46   # during the analysis
47   #
48   sample_x = None
49   sample_curves = []
50   sample_i = 0
51   sample_rate = 250
52   def sampler_cb(x, y):
53       global sample_x, sample_curves, sample_i, sample_rate
54
55       if sample_i == 0:
56           sample_x = x
57
58       if sample_i % sample_rate == 0:
59           sample_curves.append(y)
60
61       sample_i = sample_i + 1
62
63   #
64   # Run a series of analyses with varying maximum allowed order
65   #
66   #
67   # Specify the standard deviation for the move partition
68   #
69   pd = 1.0
70
71   #
72   # Run an analysis with reduced max order to only allow linear
73   # segments
74   #
75   burnin = 10000
76   total = 50000
77   max_partitions = 10
78   max_order = 3
79
80   results = rjmcmc.regression_part1d_sampled(data,
81                                              sampler_cb,
82                                              pd,
83                                              burnin,
84                                              total,
85                                              max_partitions,
86                                              max_order)
87
88   #
89   # Plot the data with black crosses, the sample curves as faint lines, and
90   # the mean as a red line
91   #
92   fig = matplotlib.pyplot.figure(1)
93   ax = fig.add_subplot(111)
94
95   yc = 0.5
96   yalpha = 1.0/((1.0 - yc) * float(len(sample_curves)))
97   for sy in sample_curves:
98
99       ax.plot(sample_x, sy,
100              color = str(yc),
101              alpha = yalpha,
102              linestyle = '-',
103              linewidth = 10)
104
105  ax.plot(results.x(), results.mean(), 'r-')
106  ax.plot(x, y, 'ko')
107  ax.set_xlim(xmin, xmax)
108
109  fig.savefig('ch4-confidence.pdf', format='PDF')
110
111  fig = matplotlib.pyplot.figure(2)
112  ax = fig.add_subplot(111)
```

```
113
114   ax.plot(results.x(), results.mean(), 'r-')
115   ax.plot(x, y, 'ko')
116   ax.plot(results.x(), results.credible_min(), 'b:')
117   ax.plot(results.x(), results.credible_max(), 'b:')
118
119   ax.set_xlim(xmin, xmax)
120
121   fig.savefig('ch4-confidenceintervals.pdf', format='PDF')
122
123   matplotlib.pyplot.show()
```

The resulting plot is shown in Figure 7. This plot shows the mean as well as 200 curves sampled from the fitting process overplotted with transparency so that the darker regions are where the true curve is more likely to be. What should be noted is that there are large spikes about the discontinuities and this is to be expected as the fitting process will trial fitting curves across these discontinuities which will produce high order curves which will overshoot.

Another feature to notices is that at approximately x = 4.5, the curves tends to take 2 alternate paths. This type of feature is likely due to whether a partition boundary is located at approximately x = 5 or not.

An alternative look at the breadth of curves is to plot the confidence intervals that are generated during the fitting process. This result is shown in Figure 8.

The confidence interval curves are shown in blue dashed lines. What these curves represent is 95 percent of the curves generated during the fitting process were contained within these lines. Similarly to the sampled curves, there is some overshoot near severe discontinuities. Lowering the maximum order can sometimes alleviate these.
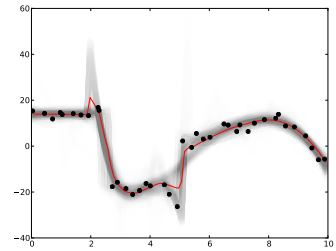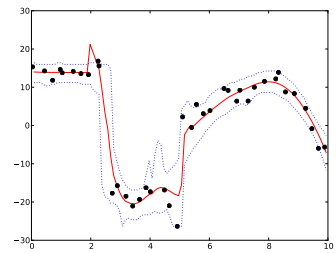

Figure 7: Sampled Curves from the Fitting

Figure 8: Confidence Intervals from the Curve Fitting