Obtaining rates of glacial isostatic adjustment from unequally spaced data   
  
John Lawson  
 University of Waterloo Earth Sciences Honours Thesis

### Contents

1 [Abstract](#x1-20001)   
2 [Introduction](#x1-30002)   
3 [Previous Work](#x1-40003)   
4 [Methods](#x1-50004)   
5 [Discussion](#x1-60005)   
5.1 [GIA Calculation Results](#x1-70005.1)   
5.1.1 [ATB-BATB](#x1-80005.1.1)   
5.1.2 [TAHB-BATB](#x1-90005.1.2)   
5.1.3 [TAHB-ATB](#x1-100005.1.3)   
5.1.4 [GTB-BATB](#x1-110005.1.4)   
5.1.5 [GTB-ATB](#x1-120005.1.5)   
5.1.6 [GTB-TAHB](#x1-130005.1.6)   
5.1.7 [Complete Table of GIA Rate comparisons](#x1-140005.1.7)   
6 [References](#x1-150006)   
7 [Appendix](#x1-160007)   
7.1 [Source code for giaModel.py](#x1-170007.1)   
7.2 [Source code for giaUtils.py](#x1-180007.2)   
7.3 [Source code for rawhide/bootstrapper.py](#x1-190007.3)   
7.4 [Source code for rawData.py](#x1-200007.4)   
7.5 [Source code for dataModel.py](#x1-210007.5)   
7.6 [Source code for linearInterpolationModel.py](#x1-220007.6)

### 1 Abstract

The ground surface underlying the Laurentian Great Lakes is currently undergoing vertical adjustment after being depressed by the weight of an ice sheet formed in the most recent glacial period during the Wisconsonian. The rate of glacial isostatic adjustment (GIA) varies by location, and influences the flow of water in the Laurentian Great Lakes (LGL) as the inclination of the ground surface differentially changes. Previous attempts to estimate the rate of GIA between sites used water gauge data from the past ~150 years in order to measure the rate of this long-term geologic process. By inferring GIA from measurements of past water levels preserved in the geological record over the past 5000 years provides a more accurate estimate of the long-term geological process of GIA. These data include measuring the elevation of a subsurface sedimentary contact relating to past lake levels, which are then age-dated using optically stimulated luminescence (OSL) to provide an age for sediments and a specific lake stage. Elevation and age data are then compiled to create site paleohydrographs for each location around the lake basin that are compared to resolve GIA around the lake basin.

The focus of this paper is to analyze the elevation and age data compiled by Johnston et al, 2012 in shorelines of Lake Superior to interpret rates of GIA between sites that have unequally spaced data. To deal with unequally spaces data each sites Paleohydrographic data is first linearly interpolated between measured data points, then subtracted between sites from one measured elevation from one site a linearly modelled elevation of another site to create a plot of difference in relative elevation over time. Once this is done, the rate of change per unit time is obtained from a linear regression though elevation differences, interpreted to represent the rate of GIA between each pair of sites. This process is repeated for all six combinations between four study sites of Johnston et al. (2012) in Lake Superior, namely Grand Traverse Bay (GTB), Au Train Bay (ATB), Batchawana Bay (BATB), and Tahquamenon Bay (TAHB). The results of this process were in good agreement at the 95% confidence level for GIA rates obtained from forward and reverse regressions for the combination of ATB-BATB (23.5 to 31 cm/century) and BATB-TAHB (11 to 17 cm/century). Agreement was also seen at the 95 % confidence level for GTB-TAHB (anywhere from -3 to 8.5 cm/century), ATB-GTB (9 to 13 cm/century) and ATB-TAHB (19.5 to 29 cm/century).

### 1 Introduction

The Earths crust rests on top of the mantle, its elevation rising and falling with the amount of mass weighing on it. During glacial periods, a significant portion of the water on earth is transferred in form from water in the oceans to glacial ice sheets, weighing down the continental crust and causing the mantle to dynamically adjust with it. This causes the crust to ride relatively lower in elevation, a change which reverses when the weight is removed as the ice sheets melt. This vertical motion of the crust while attempting to return to its previous position is known as glacial isostatic adjustment (GIA) (Scott et al, 2010).

The process of GIA has implications for the routes that the flow of water on the Earths surface takes; the ”tilting” of the surface caused by uneven rates of GIA in different locations (the thickness and residence time of the ice sheet impacts how much the crust subsides and how rapidly it rebounds) may open or close drainage outlets from basins, causing some rivers and lake outlets to go dry, while opening new outlets for water to flow through as the ’tilt’ of the lake basin changes and some locations rise and fall in elevation. Additionally, the change in ”tilt” has potential to change shorelines of existing basins, which has implications for land usage and long term engineering projections for structures such as locks and dams, where the lifetime of the structure may extend over decades or centuries. In the case of locks providing access to a canal, a local change in water level on the order of centimeters to meters over a century could cause the structure to become submerged below the rising water level, making the structure obsolete well within its intended usage lifetime.

### 2 Previous Work

Mainville & Craymer (2005) used water gauge data collected around the LGL over the past 150 years to create monthly means of water level. Differences in these values between sites were then plotted against time to calculate a rate of elevation change between sites over time. This value is interpreted to represent the long-term process of GIA in the LGL, even though the actual process extends over a much longer timescale. Combinations of sites were shown to produce inconsistent results, so a second method using a least squares adjustment process was used, repeatedly removing some monthly mean outliers which plotted a residual distance away from the linear regression. A third, and ultimately optimal method for calculating GIA was developed by Mainville & Craymer in their 2005 paper, using the original method of directly comparing monthly water level means, but this time with adjustments for the epoch, site, and month of the year. Their findings with this method showed a general agreement with the post glacial ICE-3G global model of GIA at that time, while (Mainville & Craymer, 2005).  
  
Johnston et al. (2012) attempted to provide a value for GIA in the LGL with better accuracy than previous estimates using water gauge data. In order to accomplish this, the data used to measure the process of GIA needed to extend over a much longer timescale. In this method, water levels were inferred from the elevation of relict shorelines in beach ridge strandplains from the late Holocene sediment record surrounding Lake Superior. Ages for each elevation were inferred from age-dating beach deposits or beach ridges in strandplain sequences using optically stimulated luminescence (OSL). Johnston et al. (2012) and Mainville & Craymer (2005) data differed because elevations were not and were sampled at the same points in time, respectively. Therefore, Johnston et al. (2012) calculated individual regression lines per site for a series of four ranges of time related to lake level phases, namely Nipissing, Algoma, Sault and Sub-Sault. The results reported from this are summarized in Figure [1](#x1-4001r1).

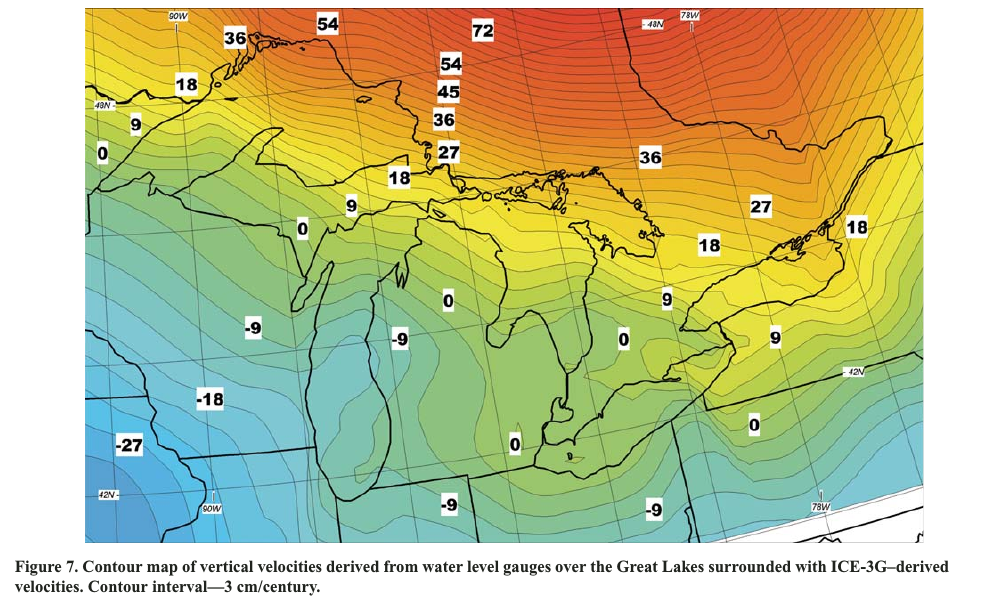


Figure 1: GIA values reported by Johnston et al 2012. All values are in cm/century.

The rate of GIA is estimated by comparing the elevation of the water level at two different locations around a common basin, and calculating this difference over a certain time period that is long enough to represent the relatively slow process of GIA, at least 3 to 4 decades using historical records or more accurately using longer geological records. The most accurate elevation of the water surface has been measured inside ancient shorelines called beach ridges (Johnston et al. 2014). These beach deposits are preserved in embayments forming strandplain sequences.Their ages have been determined by optically stimulated luminescence (OSL) dating sediment directly in beach ridges close to where elevations are measured. Elevation and age data for ?? number of beach ridges in four strandplains adjacent to Lake Superior are shown in Figure [2](#x1-4002r2). Add text description of sites…

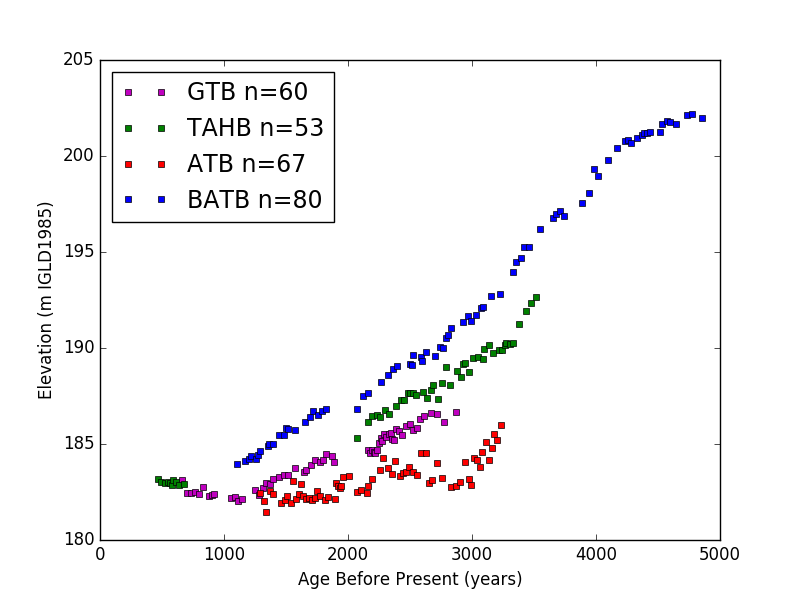


Figure 2: Current day elevation of relict shorelines with respect to time before present over the last 5000 years. Strandplain sites Au Train Bay, Michigan (ATB), Grand Traverse Bay, Michigan (GTB), Batchawana Bay, Ontario (BATB), and Tahquamenon Bay, Michigan (TAHB) surrounding Lake Superior are plotted individually. Data from Johnston et al. (2012)

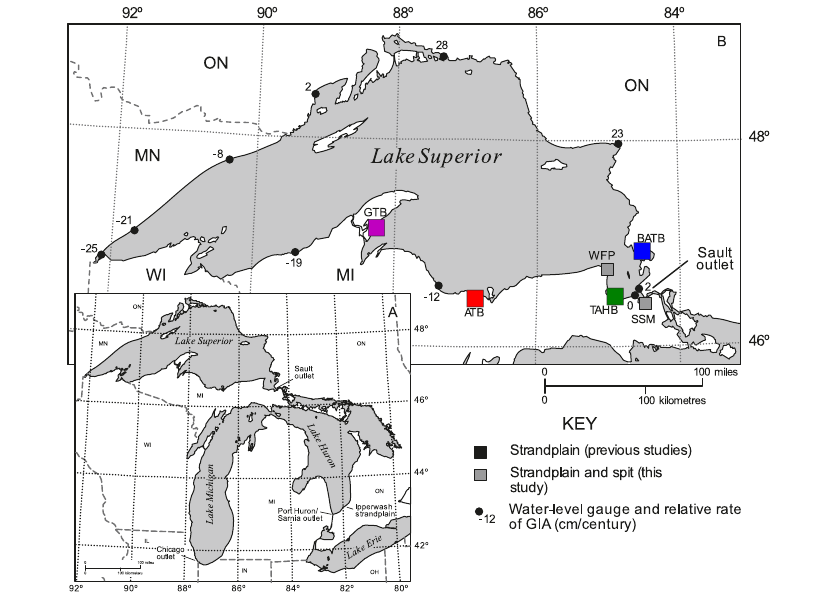


Figure 3: Map of the Upper LGL, showing locations of the modern Sault and Port Huron outlets as well as the ancient Chicago outlet. Strandplains have been colour coded to help keep track of four different study sites (i.e. blue is BATB, green is TAHB, purple is GTB, and red is ATB (modified from Johnston et al., 2012).

Observing Figure 2?, it can be seen that all four datasets follow somewhat linear trends, decreasing in elevation as the time before present approaches the current day. Johnston et al. (2012) interprets this long-term pattern of relative water level change as GIA and the short-term pattern of relative water level change as climate. In other words, older shorelines have rebounded further upwards over the past 5000 years than the younger shorelines, creating an apparent lowering in elevations across the strandplain towards the modern Lake Superior shoreline. And for strandplains that are further northeast are elevated more than strandplains that are further southwest. This can be explained by the closer proximity to the ancient center of the Laurentide Ice Sheet, roughly located near current-day James Bay in northern Ontario that experienced thicker ice and longer duration of ice but is still readjusting since the ice sheet melted.

The data within each strandplain of Johnston et al. (2012) varies. Between the four study sites, the most common feature is good data coverage between approximately 1000 and 3500 years ago at all sites, and a common gap in coverage around 2000 ago in three of the four sites. The gap in data is related to a relative low water-level time period after the Algoma highstand during a millennial lake level fluctuation (Johnston et al, 2014). While BATB has data coverage that extendsback to 5000 years ago, the other datasets do not, which makes relative GIA comparisons impossible over that time range.   
In order to measure a relative rate of GIA between sites, the rate at which these trends diverge must be measured. In the previous work of Johnston et al. (2012) using this dataset, GIA was calculated by subtracting linear regressions between study sites over the age range of each lake phase (i.e. Algoma). This was an effective first approximation of GIA, but failed to take into account the unique nature of each dataset, at times diverging from a linear relationship over each lake phase. Although these short-term variations may better relate to climatic variations, one must better compare these datasets to extract the most accurate rates of GIA.

In order to better estimate GIA from ancient shorelines one would need to simply subtract the differences in elevation between sites and plot these differences with respect to time, similar to the method used by of Mainville & Craymer (2005) with water level gauge data. Unfortunately however, none of the datasets of Johnston et al. (20012) have elevations of the same times. In other words, when one dataset has a data point present, the other dataset does not have a data point present. The objective of this new research presented hereis to develop a novel method of calculating differences in elevation between data points and modelled data bewteen sites for the strandplain paleohydrographs published in Johnston et al (2012). Calculated rates of GIA will then be compared to rates of GIA calculated in prior studies using geological data (Johnston et al. 2012) and historical data (Mainville and Craymer 2005).

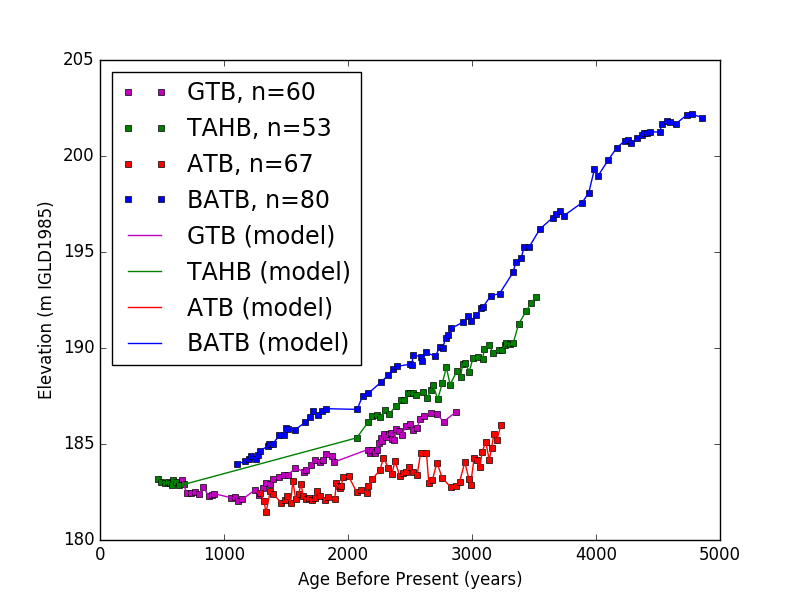


Figure 4: Strandplain paleohydrographs showing current day elevation of relict shorelines with respect to time before present over the last 5000 years. Linear interpolation applied between data points, known in this paper as modelled data

### 3 Methods

The data used in this new research is from Johnston et al. (2012). Elevations were measured in cores through beach ridges to record subsurface elevations representative of past lake level elevations. Ages for each beach ridge cored was calculated from OSL ages collected across strandplains. To account for unequally spaced elevation data between strandplains modelled elevations were calculated for each site where elevation was not directly measured by using linear interpolation between data points. This is shown as a solid line between points for each strandplain in Figure [4](#x1-4004r4). Once this estimate of elevation for times between sampled datapoints at each site was created, the difference in elevation between sites were calculated by subtracting the elevation of a measured data point from the modelled elevation of another dataset at that point in time. An example difference in elevation between strandplains is shown as a dashed line in Figure [5](#x1-5001r5).

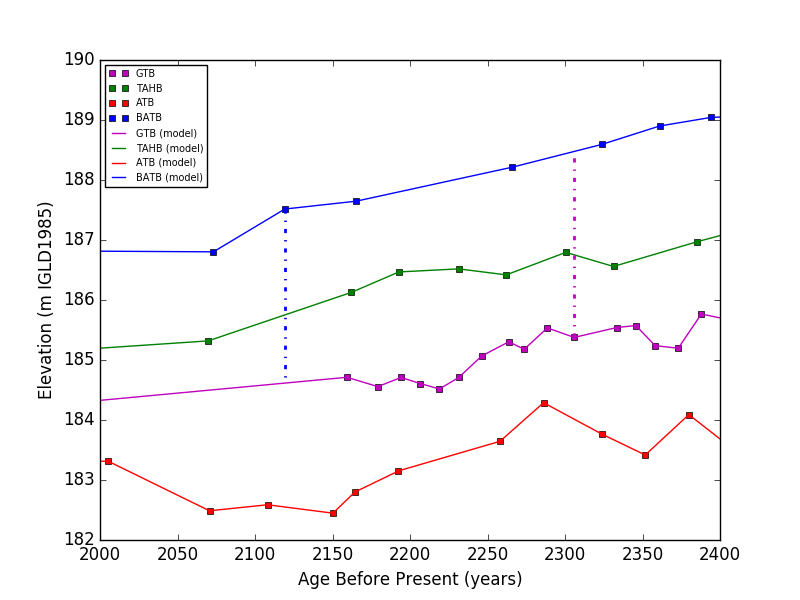


Figure 5: Example GIA comparison between strandplain datasets. Comparison made from a data point to the linear interpolation model, represented by a vertical alternating dashed line

To avoid making comparisons between data in one dataset and modelled data in another dataset in areas where the model extends over long time periods between measured data (i.e. 1500 year gap in TAHB shown in Fig. ??), the data was grouped. All four sites were grouped into a series of bins starting at 450 years before present with a width of 200 years for each bin. The start point was chosen by taking the youngest age value recorded in any of the datasets and rounding down to the nearest ten, in this case 450 years before present. The 200-year bin width or maximum acceptable age duration to compare between strandplains was chosen because it corresponds to the average error of strandplain age models published in Johnston et al (2012). If any bin had no data available for one site or the other site while comparing strandplain records, none of the data points in that bin range were used to make comparisons, thus ensuring that areas like the long gap in TAHB were not included when calculating GIA. In addition, a second rule was created, stipulating that the counts for each bin needed to be within 75% of one another to be considered in calculating relative elevation between sites. This helped identify a few areas where the datasets for both bins compared poorly, but produced valid comparison windows. An example is shown in figure [4](#x1-4004r4) where these is some overlapbetween GTB and TAHB between 650 and 450 years before present.  
In order to implement the methods described above, that include subtracting measured and modelled elevations while accounting for gaps and ample data a python script was written to analyze the data of Johnston et al. (2012). The source code used for this thesis is in the Appendix.

Using comparisons from measured data points to linear interpolation model for pairs of study sites, two graphs of the relative difference over time was created for each pair of sites. For example, for the site combination of ATB & BATB, ATB was first compared to BATB, followed by a comparison from BATB to ATB. The rates of GIA produced by these comparisons should be of opposite signs, but similar magnitudes.

The GIA rates are determined by applying a linear regression to each comparison, the slope of each regression representing the relative rate of GIA between sites.

### 4 Results and Discussion

5.1 GIA Calculation Results

Listed in this section are the results of each of the possible combinations of sites. Since 4 sites were used, a total of 6 distinct combinations of sites were studied.

5.1.1 ATB-BATB

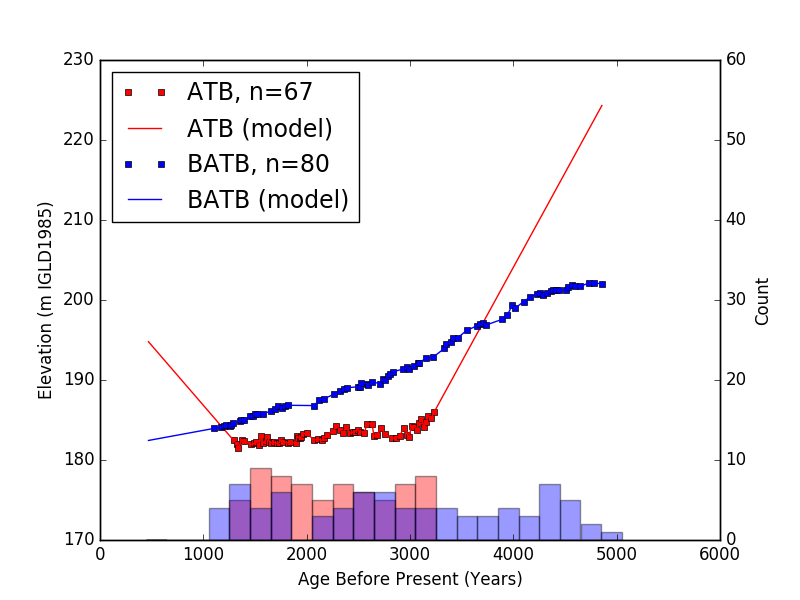


Figure 6: Measured and modelled elevation data plotted against age for sites ATB & BATB. Data grouped into bins with widths of 200 years starting at 450 years before present. Bin Counts shown as a histogram at bottom of graph, one each for ATB, BATB and both ATB and BATB.

The data available for sites ATB and BATB shows two of the most common trends in the data used in this paper; Data is available for both sites from approximately 1000 to 3300 years before present, with a gap in the record at around 2000 years before present. With the data divided up into bins of 200 years width starting at 1050 years before present, the data from every bin between 1250 and 3250 was used in calculating a rate of GIA, save for the gap from 1850-2050 years before present caused by the Algoma low water level (where comparisons between data in the ATB dataset would be subtracting against a modelled value for BATB that is unreliable given the distance to the nearest datapoint in BATB). The regressions derived from this pair of data sets, seen in Figures [8](#x1-8003r8) & [9](#x1-8004r9), are well constrained, and produce a value on relative GIA between ATB and BATB of 24.7 - 31.0 cm/century. In order to produce this range, a regression was done on both comparisons, the results of which are reported in Figure [7](#x1-8002r7). The absolute rate of GIA was determined from the absolute value of the slopes of each regression, reported here as a 95% confidence interval in cm/century. Once these 95% intervals were created, a final value was created from the range where both confidence intervals overlapped. (for example, with this site the two ranges are -24.71 to -31.01 and 23.51 to 31.45 cm/century. Converting to absolute value, the overlap between (24.71,31.01) & (23.51,31.45) is the range (24.71,31.01) ). A complete plot of the confidence intervals for the slopes obtained from every linear regression done in this paper can be seen in Figure [30](#x1-14001r30).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | | | |  |  |  |
| name | Slope Estimator | | | | | Slope Error | r Squared | Slope C.I. (95p) |
|  |  | | | | |  |  |  |
| ATB relative to BATB model | -27.86039 | | | | | 1.60824 | 0.860 | -24.70824 to -31.01254 |
| BATB relative to ATB model | 27.48266 | | | | | 2.02672 | 0.840 | 31.45503 to 23.51028 |
|  |  | | | | |  |  |  |
|  |  |  |  |  |

Figure 7: ATB-BATB Linear regression output parameters.

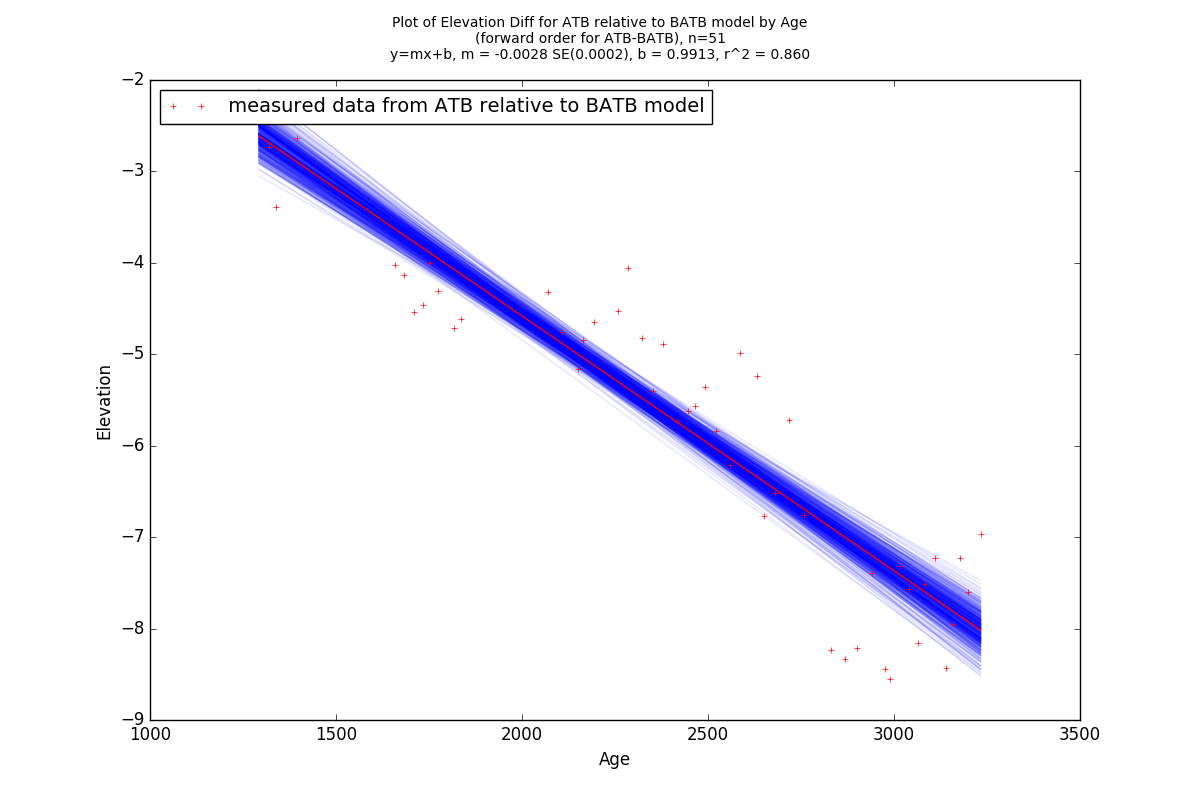


Figure 8: Differences in elevation from ATB measured data to BATB modelled data. 95p Bootstrap of the main regression rendered in blue around the estimator version of the regression (rendered as a solid red line).

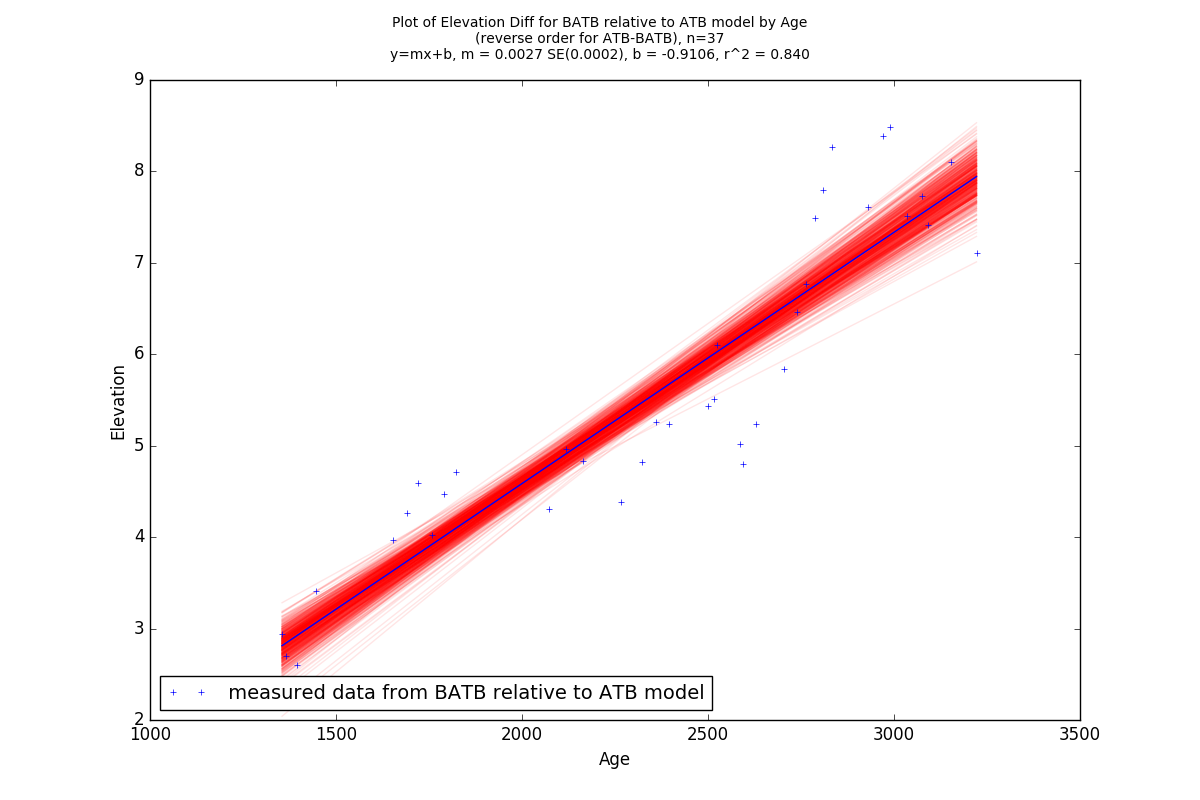


Figure 9: Differences in elevation measured from the BATB data to the ATB model. 95p Bootstrap of the main regression rendered in red around the estimator version of the regression (rendered as a solid blue line).

5.1.2 TAHB-BATB

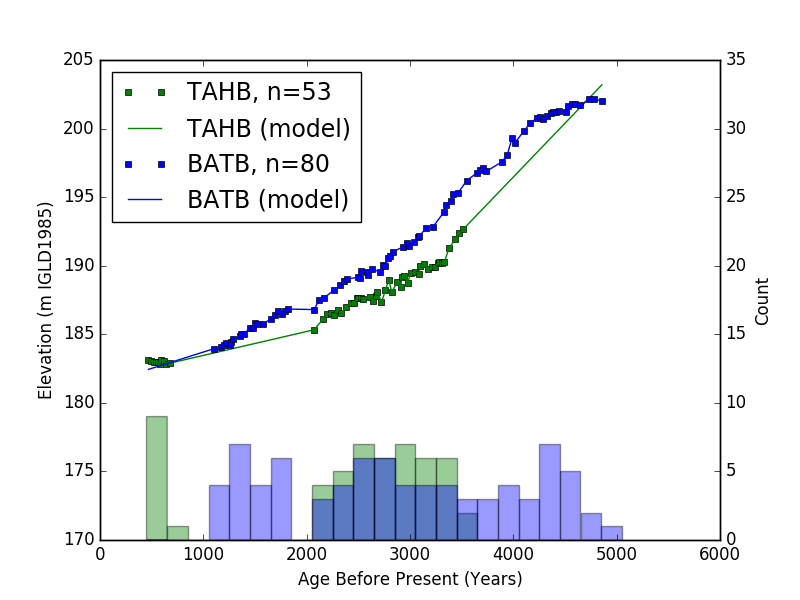


Figure 10: Measured and modelled elevation data plotted against age for sites TAHB & BATB. Data grouped into bins with widths of 200 years starting at 450 years before present. Bin Counts shown as a histogram at bottom of graph.

The data plot for the site combination of TAHB and BATB shows a common issue with comparing datasets, as the data with ages more recent than the 2000 year before present gap is unusable. This is because the regions where data is available for one dataset are empty of datapoints for the other, making the modelled prediction of the other dataset unreliable. As a result, a filter is applied to the data to prevent this, grouping data points into bins 200 years wide starting at 450 years before present, and ignoring the data points from bins in which either data set had no datapoints, as well as any which had bin counts differing by more than 75% for that bin. As a result, only the data from 2050 to 3650 years before present were used in creating the GIA comparisons between TAHB and BATB.  
The linear regressions produced from this pair of datasets are shown in Figures [12](#x1-9003r12) & [13](#x1-9004r13), with the parameters for each regression listed in Figure [11](#x1-9002r11). Merging the two ranges reported under the ”Slope C.I. (95p)” column in Figure [11](#x1-9002r11), a value for relative GIA of between 11.9-16.8 cm/century is seen.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | | | |  |  |  |
| name | Slope Estimator | | | | | Slope Error | r Squared | Slope C.I. (95p) |
|  |  | | | | |  |  |  |
| TAHB relative to BATB model | -14.32814 | | | | | 1.24125 | 0.765 | -11.89530 to -16.76099 |
| BATB relative to TAHB model | 14.00018 | | | | | 1.54265 | 0.733 | 17.02377 to 10.97660 |
|  |  | | | | |  |  |  |
|  |  |  |  |  |

Figure 11: TAHB-BATB Regression output parameters

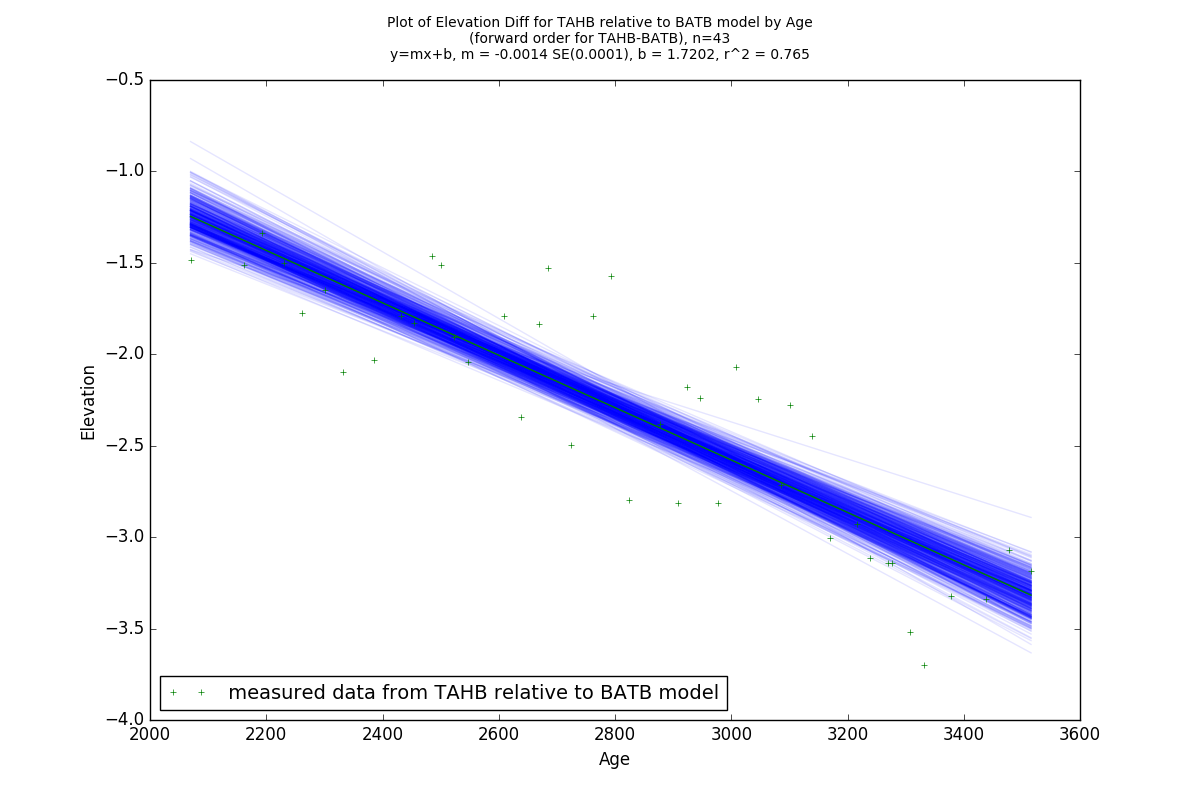


Figure 12: Differences in elevation measured from the TAHB data to the BATB model. 95p Bootstrap of the main regression rendered in blue around the estimator version of the regression (rendered as a solid green line).

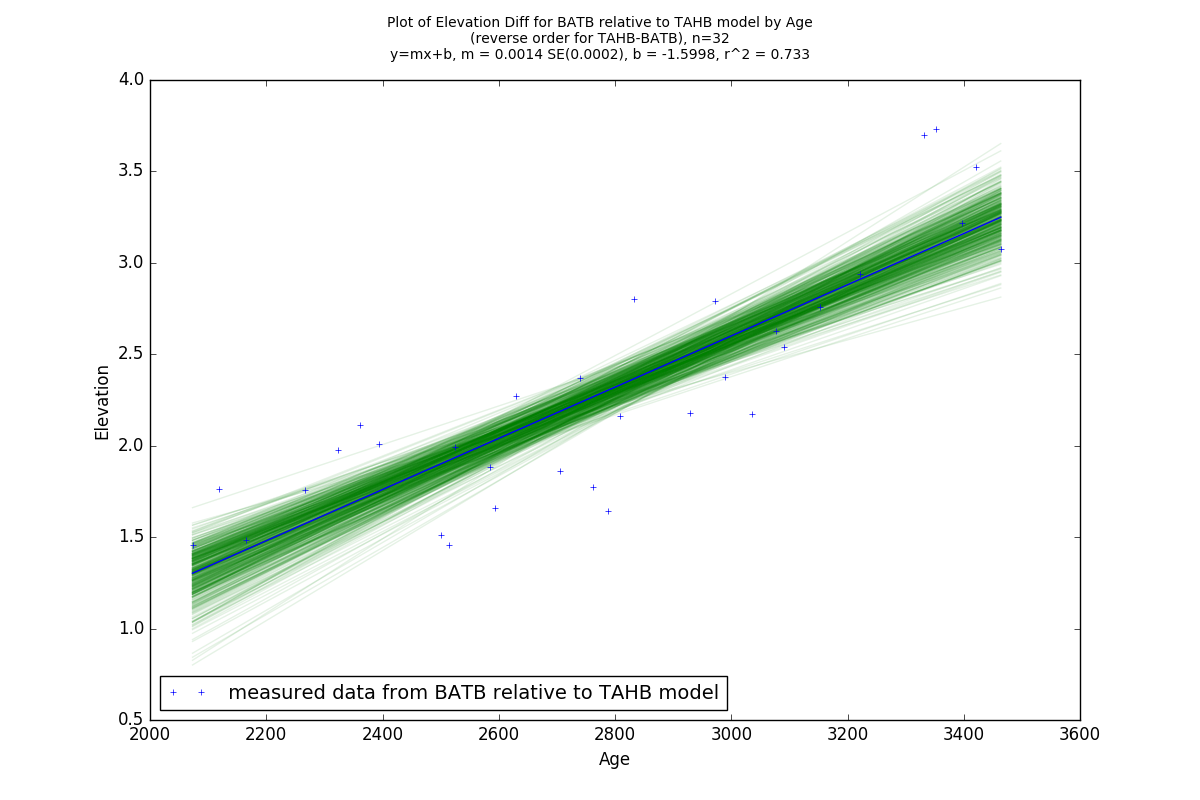


Figure 13: Differences in elevation measured from the BATB data to the TAHB model. 95p Bootstrap of the main regression rendered in green around the estimator version of the regression (rendered as a solid blue line).

5.1.3 TAHB-ATB

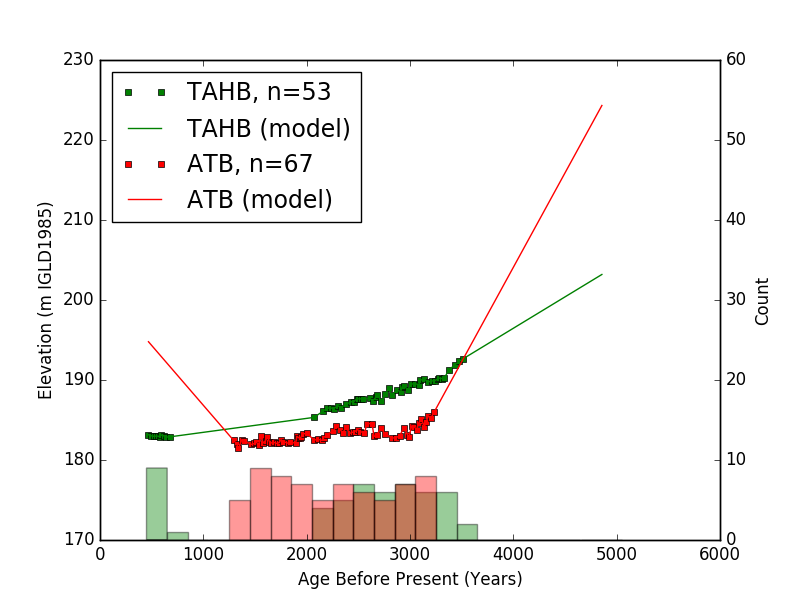


Figure 14: Measured and modelled elevation data plotted against age for sites TAHB & ATB. Data grouped into bins with widths of 200 years starting at 450 years before present. Bin Counts shown as a histogram at bottom of graph.

Similar to previous dataset comparisons TAHB-BATB and ATB-BATB, the combination of TAHB and ATB are constrained to ages older than 2050 years before present, but also have a much shorter range of age values that can be considered for GIA calculation, starting at 2000 and ending at around 3100 years before present. This is due to TAHB having no data available between 1250-2050 years before present, while ATB has a great deal of data in this range that can not be considered for this comparison. As a result, only datapoints between 2050 and 3250 years before present were used, resulting in relatively poor regressions (R2 values close to 0.5 where TAHB-BATB and ATB-BATB were both well above 0.7) reported in Figure [15](#x1-10002r15). The weak correlation of these regressions results in a wide range for relative GIA of between 19.4-29.2 cm/century.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | | | |  |  |  |
| name | Slope Estimator | | | | | Slope Error | r Squared | Slope C.I. (95p) |
|  |  | | | | |  |  |  |
| TAHB relative to ATB model | 26.20553 | | | | | 3.45025 | 0.643 | 32.96802 to 19.44304 |
| ATB relative to TAHB model | -23.06696 | | | | | 3.14849 | 0.599 | -16.89592 to -29.23801 |
|  |  | | | | |  |  |  |
|  |  |  |  |  |

Figure 15: TAHB-ATB Regression output parameters

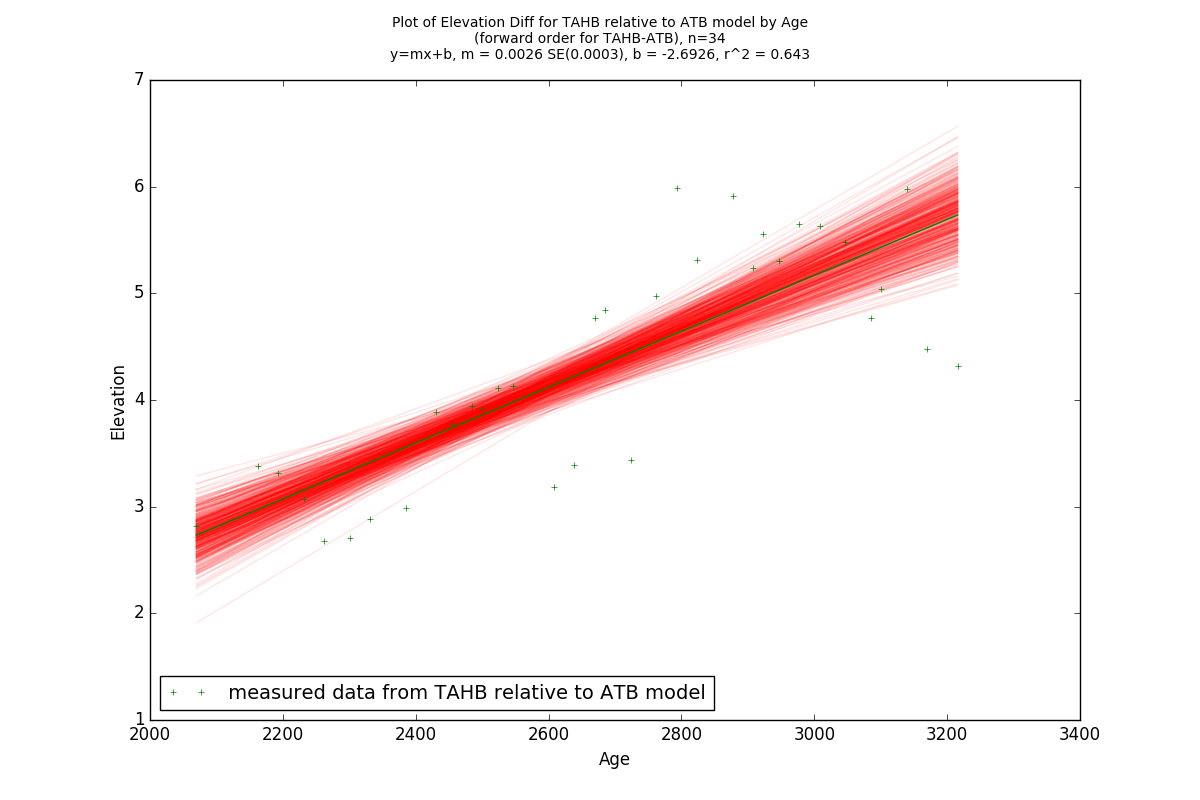


Figure 16: Differences in elevation measured from the TAHB data to the ATB model. 95p Bootstrap of the main regression rendered in red around the estimator version of the regression (rendered as a solid green line).

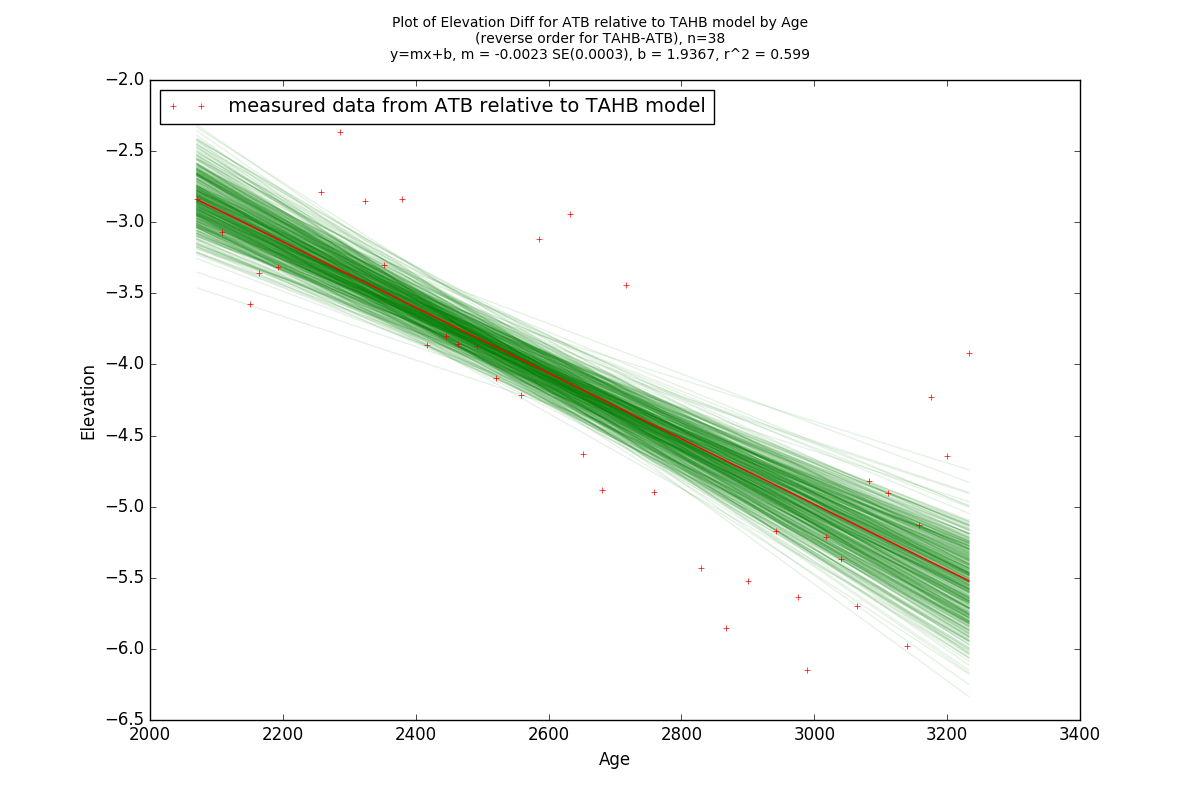


Figure 17: Differences in elevation measured from the ATB data to the TAHB model. 95p Bootstrap of the main regression rendered in green around the estimator version of the regression (rendered as a solid red line).

5.1.4 GTB-BATB

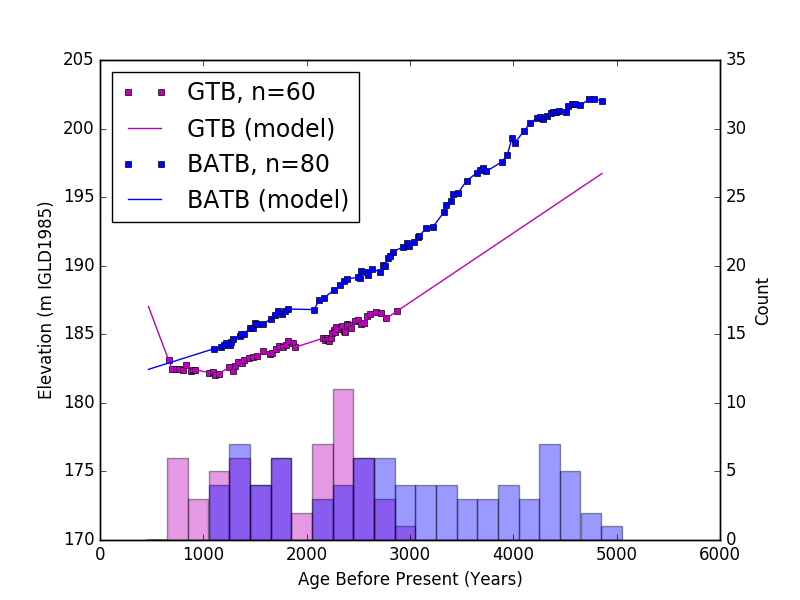


Figure 18: Measured and modelled elevation data plotted against age for sites GTB & BATB. Data grouped into bins with widths of 200 years starting at 450 years before present. Bin Counts shown as a histogram at bottom of graph.

The GTB BATB combination has data available for both datasets from 1050 to 3050 years before present with a gap in coverage from 1850 to 2050 years before present. The first case of the 75% difference cutoff rule for bin inclusion has its first appearance here as only the oldest shoreline available from GTB falls inside of the 2850-3050 years before present window, causing the entire window to be used if the only criteria was both dataset counts within that window being non-zero. The 75% cutoff prevents this window from being used in this case, as the counts for the bin at 2850-3050 years before present differ by 120% between GTB and BATB. This rule is useful in identifying areas of the dataset where both sites have data available, but the density of one of the datasets in that region is low enough to potentially cause inaccurate predictions where modelled elevation extends a long distance between measured datapoints. The regressions plotted in Figures [20](#x1-11003r20) & [21](#x1-11004r21) are listed in Figure [19](#x1-11002r19). The R2 values for both regressions are well over 0.8, and the ranges for relative GIA listed under the ”Slope C.I. (95p)” column agree to within less than 1 cm/century, producing one of the most well constrained values seen in this paper at 10.5-13.4 cm/century (a range of less than 3 cm/century).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | | | |  |  |  |
| name | Slope Estimator | | | | | Slope Error | r Squared | Slope C.I. (95p) |
|  |  | | | | |  |  |  |
| GTB relative to BATB model | -10.51109 | | | | | 0.81620 | 0.864 | -8.91134 to -12.11085 |
| BATB relative to GTB model | 12.15952 | | | | | 0.86207 | 0.865 | 13.84917 to 10.46987 |
|  |  | | | | |  |  |  |
|  |  |  |  |  |

Figure 19: GTB-BATB Regression output parameters

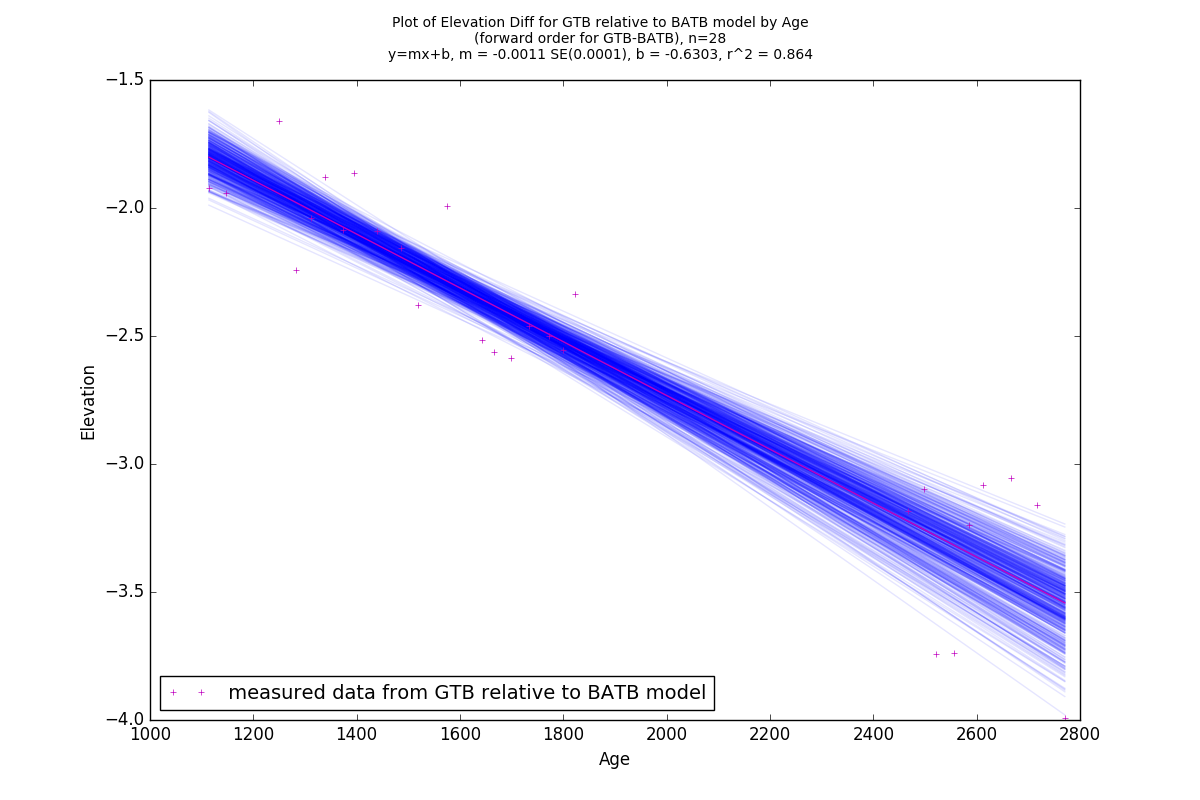


Figure 20: Differences in elevation measured from the GTB data to the BATB model. 95p Bootstrap of the main regression rendered in blue around the estimator version of the regression (rendered as a solid purple line).

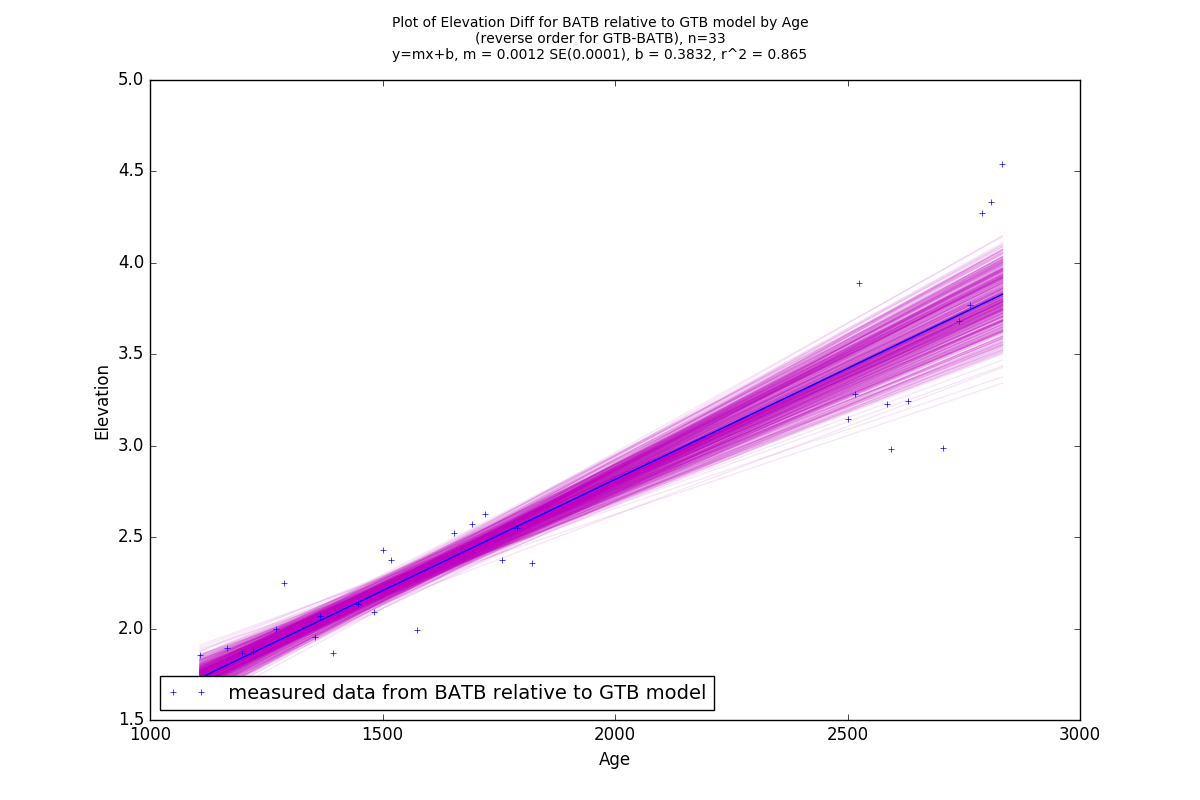


Figure 21: Differences in elevation measured from the BATB data to the GTB model. 95p Bootstrap of the main regression rendered in purple around the estimator version of the regression (rendered as a solid blue line).

5.1.5 GTB-ATB

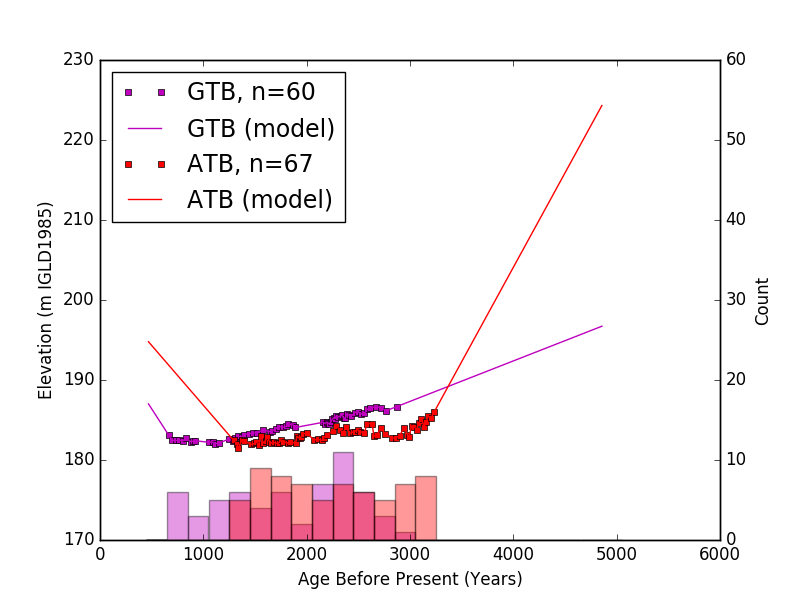


Figure 22: Measured and modelled elevation data plotted against age for sites GTB & ATB. Data grouped into bins with widths of 200 years starting at 450 years before present. Bin Counts shown as a histogram at bottom of graph.

The GTB-ATB combination has bins from 1250 to 3050 years before present containing data for both sites. Two of these bins fail to qualify for use due to the site counts differing by more than 75%. These bins can be seen at 1850-2050, and 2850 to 3050 years before present. Looking at Figure [22](#x1-12001r22), it can be seen that both of these bins coincide with ranges of time where GTB has sparse data, making the GTB models predictions unreliable in that bin. The regressions plotted in Figures [24](#x1-12003r24) & [25](#x1-12004r25), and listed in Figure [23](#x1-12002r23). These regressions have weak correlations (with R2 values of 0.427 and 0.595 respectively), and only overlap in a small range of 10.5-13.5 cm/century. Although this range appears very precise, the measurement is likely inaccurate, as the comparisons do not agree very well between the forward and reverse comparisons, the ranges of GIA produced from each regression only overlapping for a small fraction of their respective ranges at the 95% confidence level.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | | | |  |  |  |
| name | Slope Estimator | | | | | Slope Error | r Squared | Slope C.I. (95p) |
|  |  | | | | |  |  |  |
| GTB relative to ATB model | 9.58750 | | | | | 1.95903 | 0.400 | 13.42719 to 5.74780 |
| ATB relative to GTB model | -12.77402 | | | | | 1.94089 | 0.560 | -8.96988 to -16.57815 |
|  |  | | | | |  |  |  |
|  |  |  |  |  |

Figure 23: GTB-ATB Regression output parameters

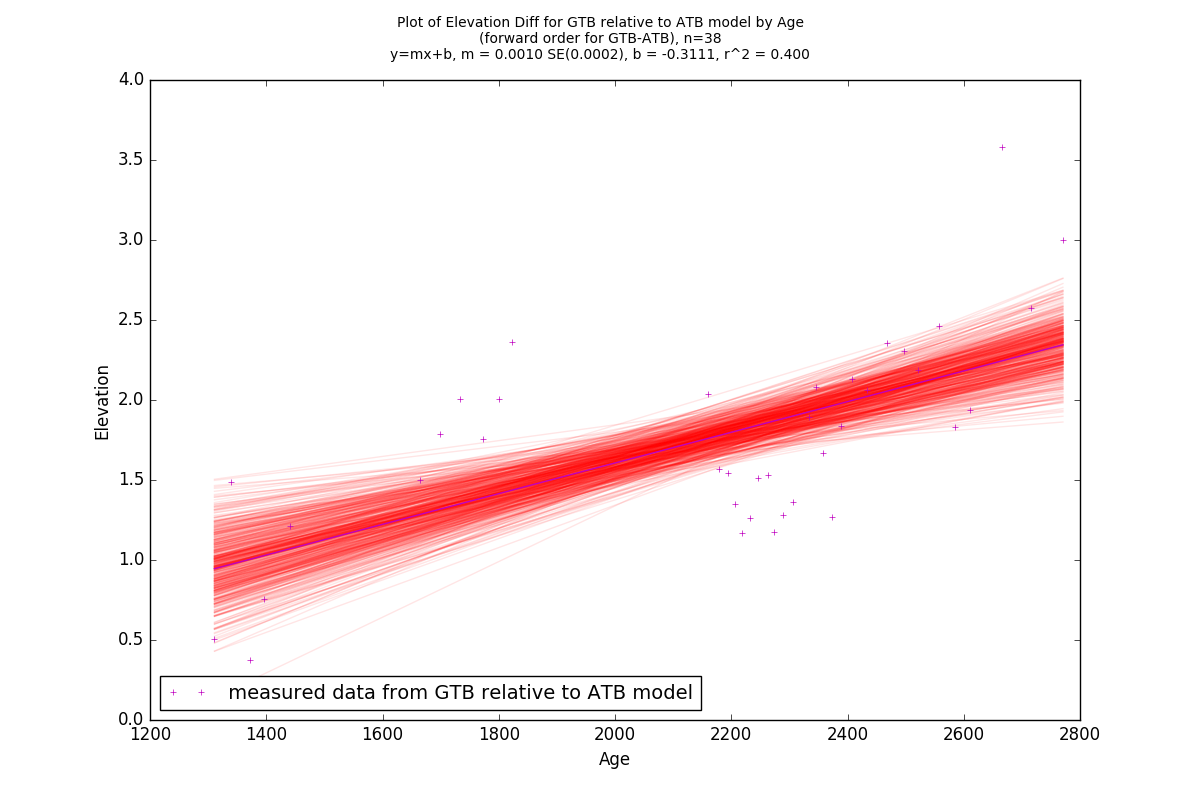


Figure 24: Differences in elevation measured from the GTB data to the ATB model. 95p Bootstrap of the main regression rendered in red around the estimator version of the regression (rendered as a solid purple line).

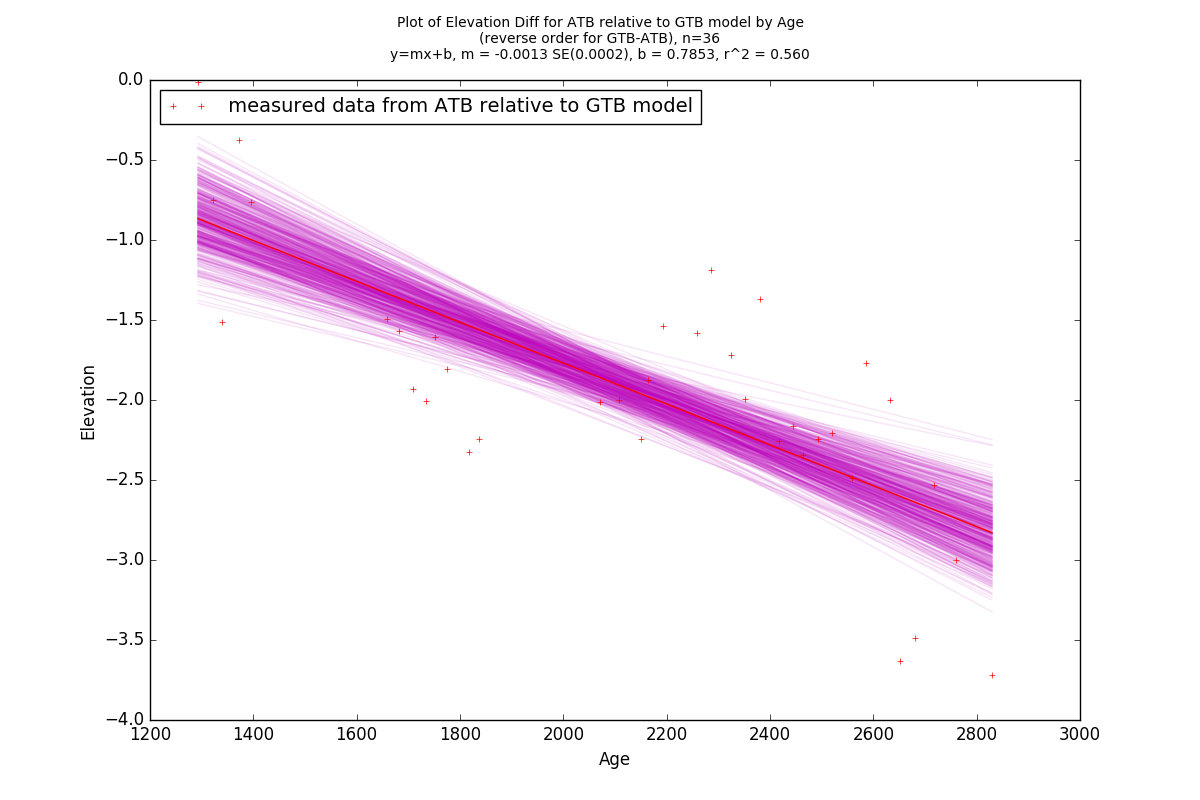


Figure 25: Differences in elevation measured from the ATB data to the GTB model. 95p Bootstrap of the main regression rendered in purple around the estimator version of the regression (rendered as a solid red line).

5.1.6 GTB-TAHB

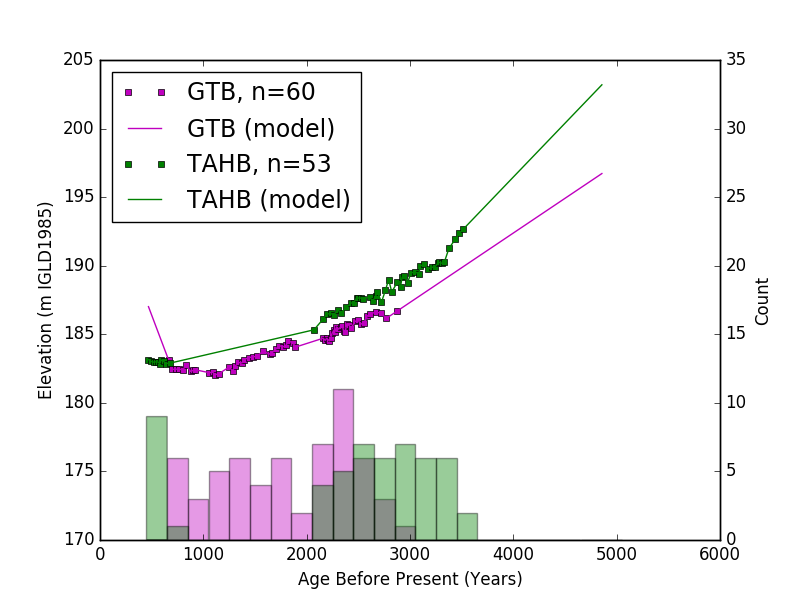


Figure 26: Measured and modelled elevation data plotted against age for sites GTB & TAHB. Data grouped into bins with widths of 200 years starting at 450 years before present. Bin Counts shown as a histogram at bottom of graph.

The combination of sites GTB and TAHB has data in bins from 450 to 3650 years before present, of which only 2050 to 2850 years before present was used. Two potential bins located at 650-850 and 2850-3050 years before present were not used due failing to meet the 75% rule, both would have produced comparisons between areas of data in one dataset and a poorly constrained model in the other.

The combination of sites GTB and TAHB has by far the poorest regressions, listed in [27](#x1-13002r27). This is likely due to the alignment of most of both datasets only giving limited sample sizes of n=22 (Figure [29](#x1-13004r29)) and n=27 (Figure [28](#x1-13003r28)). This resulted in an estimate of GIA that ranges anywhere from -2.8-8.6 cm/century, possibly implying that the vertical adjustment rates between the TAHB and GTB sites may be zero, or at the least small in comparison to other site comparisons.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | | | |  |  |  |
| name | Slope Estimator | | | | | Slope Error | r Squared | Slope C.I. (95p) |
|  |  | | | | |  |  |  |
| GTB relative to TAHB model | 1.04458 | | | | | 3.84483 | 0.003 | 8.58045 to -6.49128 |
| TAHB relative to GTB model | 5.70698 | | | | | 4.33801 | 0.080 | 14.20948 to -2.79553 |
|  |  | | | | |  |  |  |
|  |  |  |  |  |

Figure 27: GTB-TAHB Regression output parameters

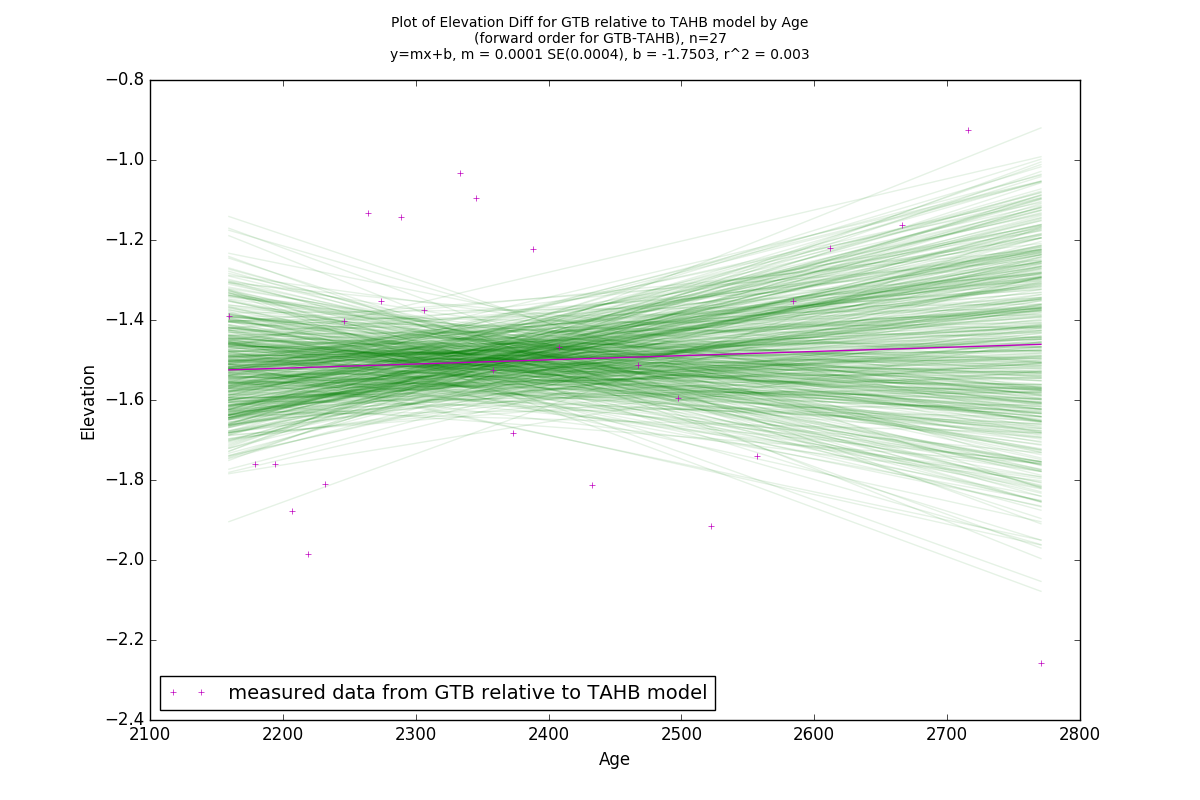


Figure 28: Differences in elevation measured from the GTB data to the TAHB model. 95p Bootstrap of the main regression rendered in green around the estimator version of the regression (rendered as a solid purple line).

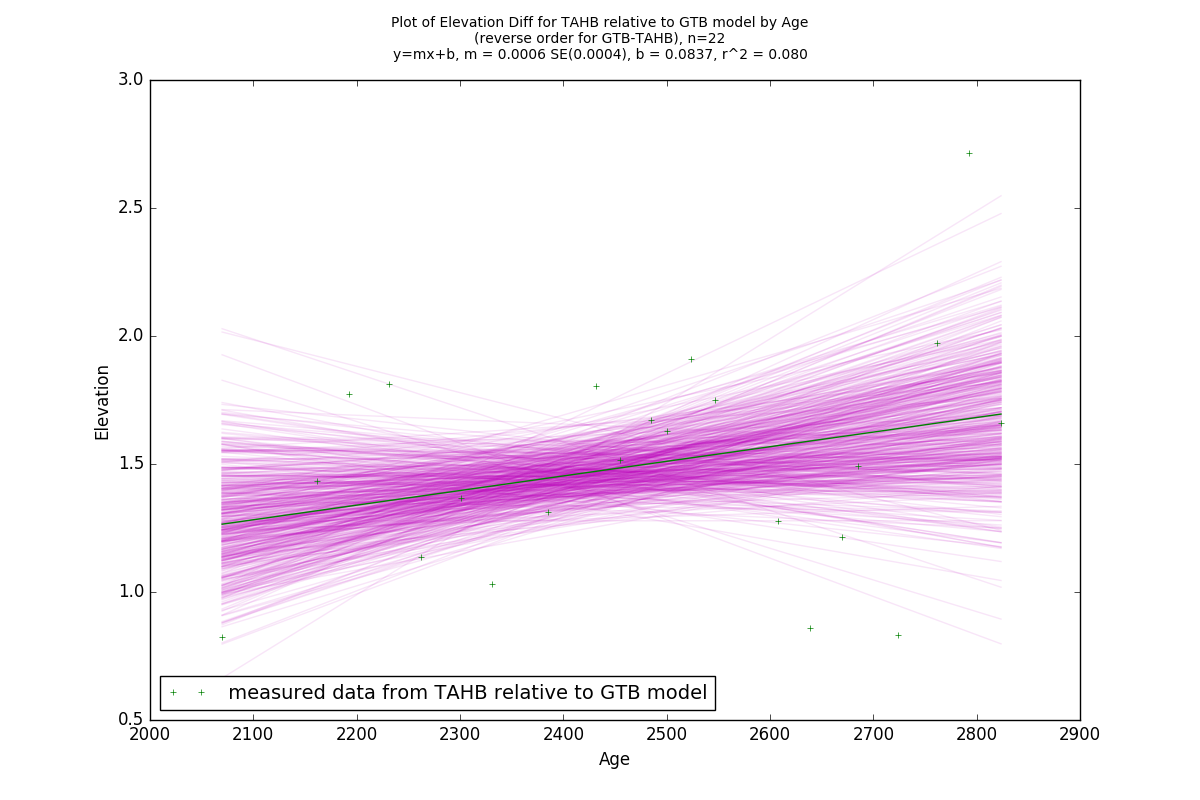


Figure 29: Differences in elevation measured from the TAHB data to the GTB model. 95p Bootstrap of the main regression rendered in purple around the estimator version of the regression (rendered as a solid green line).

5.1.7 Complete Table of GIA Rate comparisons

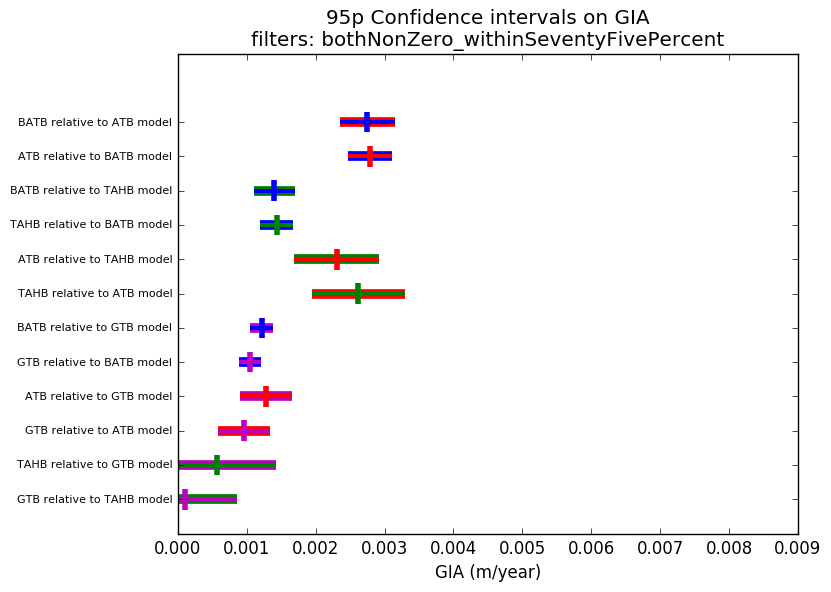


Figure 30: 95p Confidence intervals on absolute value of GIA rates obtained from 12 site comparisons across 4 sites.

In Figure [31](#x1-14002r31), the values for relative GIA produced by this paper are visualized by plotting the value range of GIA rates between each site as a line between sites on the map with the corresponding value next to it.

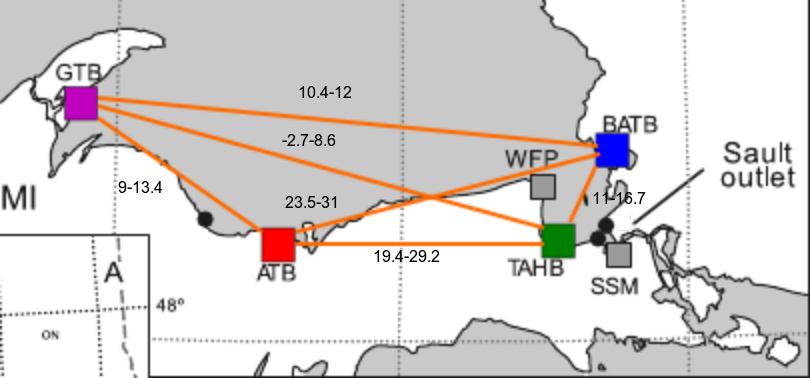


Figure 31: Relative GIA Rates produced by this papers method, all values reported in cm/century

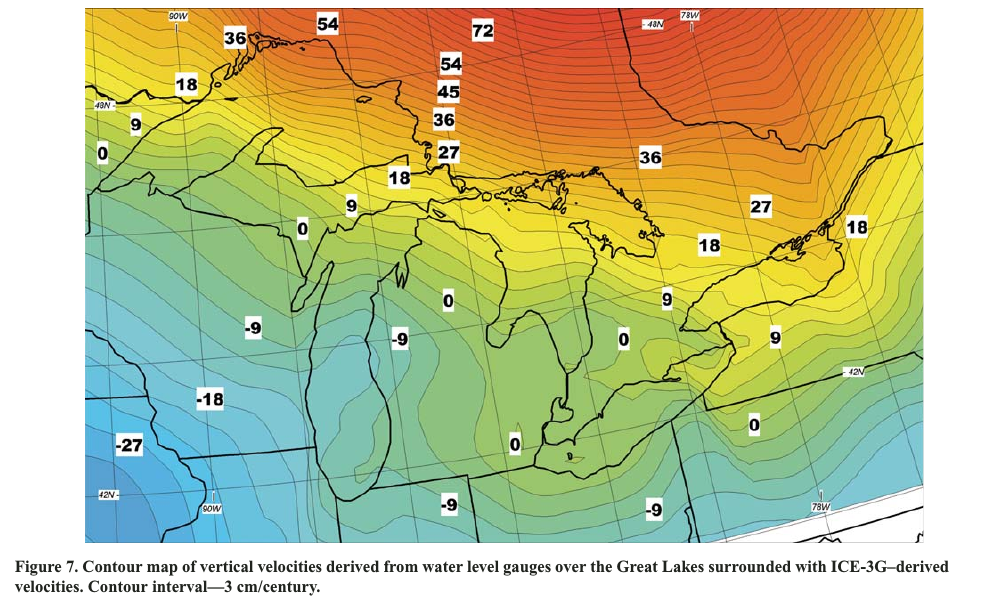


Figure 32: Relative GIA Rates produced by Mainville & Craymer, all values reported in cm/century (reproduced from Mainville & Craymer, 2005)

The equivalent values for rates between sites as produced by Mainville & Craymer are inferred from subtracting the difference in contour between sites as shown in Figure [32](#x1-14003r32), and are presented in Figure [33](#x1-14004r33).

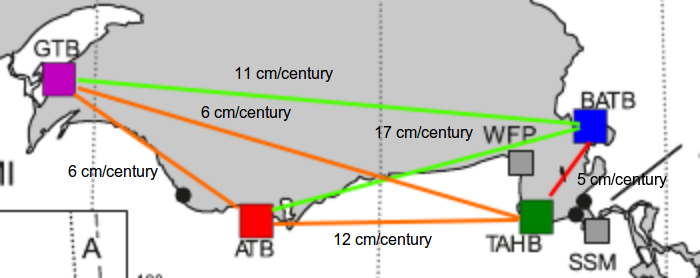


Figure 33: Relative GIA Rates produced by Mainville & Craymer

While most of the site comparisons agree reasonably well between the method employed by Mainville & Craymer and this paper, one area where significant disagreement is seen is between sites ATB, BATB, and TAHB, especially in the much larger values produced by this paper between ATB-BATB and ATB-TAHB. Given that both of these site combinations are separated by an East-West line, this could imply that the location of the center of the Laurentide Ice Sheet during the last glaciation being to the north and west of Lake Superior had a stronger effect on the overall process of rebound than the simple fact that areas to the north were more likely to be depressed by the weight of ice sheets than areas further south.

### 6 References

Mainville, A., & Craymer, M. R. (2005). Present-day tilting of the Great Lakes region based on water level gauges. Geological Society of America Bulletin, 117(7-8), 1070-1080.  
Scott, T. W., Swift, D. J., Whittecar, G. R., & Brook, G. A. (2010). Glacioisostatic influences on Virginia’s late Pleistocene coastal plain deposits. Geomorphology, 116(1-2), 175-188.  
Johnston, J. W., Argyilan, E. P., Thompson, T. A., Baedke, S. J., Lepper, K., Wilcox, D. A., & Forman, S. L. (2012). A Sault-outlet-referenced mid-to late-Holocene paleohydrograph for Lake Superior constructed from strandplains of beach ridges. Canadian Journal of Earth Sciences, 49(11), 1263-1279.  
Johnston, J. W., Thompson, T. A., & Wilcox, D. A. (2014). Palaeohydrographic reconstructions from strandplains of beach ridges in the Laurentian Great Lakes. Geological Society, London, Special Publications, 388(1), 213-228.

### 7 Appendix

7.1 Source code for giaModel.py

   
## giaModel.py #################################################################   
## attempt to model the gia between sites using the data #######################   
## in reformattedData.ods ######################################################   
################################################################################   
import pyexcel\_ods   
   
##import sys   
   
   
import csv   
   
import matplotlib.pyplot as plt   
import numpy as np   
from scipy import stats   
   
from matplotlib.font\_manager import FontProperties   
   
import itertools   
## used to generate the number of links between sites   
import random   
from random import sample   
## used in the random choice feature of the zoomed site comparisons   
   
from rawhide import bootstrapper   
## get the custom bootstrap plotting function from this projects code   
   
from linearInterpolationModel import \*   
from giaUtils import \*   
   
   
   
fontP = FontProperties()   
fontP.set\_size(’small’)   
   
   
def trendline(x, gradient, intercept):   
  ## return a y given an mx+b   
  output = gradient\*x + intercept   
  return output   
   
   
## getLinearModel: listof(Num) listof(Num) -> listof(Num) Num Num Num listof(Num) listof(Num)   
   
def getLinearModel(x\_values, y\_values, k=1.0, l=1.0):   
  gradient, intercept, r\_value, p\_value, std\_err = stats.linregress(x\_values,y\_values)   
   
  y\_model = []   
  yModelHigh = []   
  yModelLow = []   
   
  grad = k\*gradient   
  interc = l\*intercept   
   
  for x in x\_values:   
    y = trendline(x, grad, interc)   
    yHigh = trendline(x, grad+(1.96\*std\_err), interc)   
    yLow = trendline(x, grad-(1.96\*std\_err), interc)   
    y\_model.append(y)   
    yModelHigh.append(yHigh)   
    yModelLow.append(yLow)   
   
  rSquare = r\_value\*\*2   
   
  return y\_model, grad, interc, std\_err, yModelHigh, yModelLow, rSquare   
   
  ## yModelHigh and yModelLow are the y model built with a slope at the   
  ## extreme of the error bounds on the gradient   
   
   
   
   
   
   
   
   
   
   
   
   
   
   
   
   
def plotGradientConfidenceIntervals(giaRegressionsByCombo, keys, giaRegressionDescriptions, outputPathDict):   
  def plotInterval(ax, y, xstart, xstop, intervalLabel, colord, colords):   
    """Plot interval at y from xstart to xstop with given color."""   
   
    ax.hlines(y, xstart, xstop, colords, lw=7)   
    ax.hlines(y, xstart, xstop, colord, lw=3, label=intervalLabel)   
    ## plots the interval in the colours of both sites   
   
   
  outputPath = convertListToRelativePath([outputPathDict[setting] for setting in getCurrentSettingOptions()])   
   
   
   
  y = 0   
  ## used in spacing out the intervals for each site vertically through the   
  ## graph   
   
  fig,ax = plt.subplots(1)   
   
  for combo in keys:   
    y += 1   
    combo1 = combo.split(’-’)[0]   
    combo2 = combo.split(’-’)[1].split(’:’)[0]   
    order = combo.split(’-’)[1].split(’:’)[1]   
   
    if(order == ’forward’):   
      direct = combo1   
      modelled = combo2   
    else:   
      direct = combo2   
      modelled = combo1   
   
    est = giaRegressionsByCombo[combo][’gradientEstimator’]   
   
    ciStart = giaRegressionsByCombo[combo][’gradient’][0]   
    ciEnd = giaRegressionsByCombo[combo][’gradient’][1]   
   
    if(est < 0):   
      est = -est   
      ciStart = -ciStart   
      ciEnd = -ciEnd   
   
    if(order == ’forward’):   
      plotInterval(ax, y, ciStart, ciEnd, "", mapSiteToColour(direct), mapSiteToColour(modelled))   
    else:   
      plotInterval(ax, y, ciStart, ciEnd, "", mapSiteToColour(direct), mapSiteToColour(modelled))   
   
   
    ax.vlines(est, y+0.3, y-0.3, mapSiteToColour(direct), lw=4)   
  ax.set\_xlabel(’GIA␣(m/year)’)   
   
  ax.set\_xlim([0,0.009])   
   
  plt.yticks(list(np.arange(1, len(keys)+1, 1.0)), [giaRegressionDescriptions[key] for key in keys], rotation=0)   
   
  fileNameIdentifier = "\_".join([outputPathDict[setting] for setting in getCurrentSettingOptions()])   
   
  plt.title("95p␣Confidence␣intervals␣on␣GIA\nfilters:␣%s" % fileNameIdentifier)   
   
  for item in ax.get\_yticklabels():   
    item.set\_fontsize(8)   
   
  outputFilePath = filePathOnRelativePath(outputPath+"gias/", fileName=’intervals’, ext="png")   
  print "Saving␣gia␣intervals␣plot␣at␣’%s’" % outputFilePath   
  verifyPath(outputPath+"gias/")   
   
   
   
  plt.savefig(outputFilePath,bbox\_inches=’tight’)   
   
   
   
   
   
  outputFilePath = filePathOnRelativePath(outputPath+"gias/", fileName=’%s\_intervals’ % fileNameIdentifier, ext="png")   
   
  print "Saving␣gia␣intervals␣plot␣at␣’%s’" % outputFilePath   
   
  plt.savefig(outputFilePath,bbox\_inches=’tight’)   
   
  plt.close()   
   
   
   
   
def getDatasetsModelsAndObjects(filenameToLoad):   
  lookupTable = pyexcel\_ods.get\_data(filenameToLoad)   
  ## open up the excel file to get the data as a dict of 2-lists   
  locations = [’BATB’, ’TAHB’, ’GTB’, ’ATB’]   
  ## the first key for the lookupTable is the site location   
   
  datasets = {}   
   
  for loc in locations:   
    datasets[loc] = [row for row in lookupTable[loc]]   
    ## under each key is a rectangular list with two columns to each row,   
    ## the first one is elevation, the second one is age   
   
  for d in datasets:   
    print d, datasets[d], "\n\n\n"   
   
   
  datasetObjects = {}   
   
  datasetModels = {}   
   
  for d in datasets:   
    datasetObjects[d] = siteData(d, datasets[d])   
    ## build the dataset containers using the data retrieved for each site   
   
    ## note that the siteData object automatically filters the data received   
    ## to get rid of the first few non data lines and any empty spaces   
   
   
   
  return datasets, datasetModels, datasetObjects   
   
   
   
if(\_\_name\_\_ == "\_\_main\_\_"):   
   
  datasets, datasetModels, datasetObjects = getDatasetsModelsAndObjects("./reformattedData.ods")   
   
  allAgesSampled = [datasetObjects[d].getAgeValues() for d in datasets]   
  allAgesSampled = [item for sublist in allAgesSampled for item in sublist]   
  ## flatten out the 2-list with some list comprehension   
  print min(allAgesSampled), max(allAgesSampled)   
   
  ## create the raw plot of data points ######################################   
  for site in datasetObjects:   
    print site, datasetObjects[site].data   
    x = datasetObjects[site].getAgeValues()   
    y = datasetObjects[site].getElevationValues()   
   
    n = len(datasetObjects[site].getAgeValues())   
   
    plt.plot(x, y, mapSiteToColour(site) + ’s’, label=site+"␣n=%i" % n, markersize=4.0)   
   
    datasetModels[site] = siteModelConnectTheDots(datasetObjects[site])   
   
  ##plt.title("Plot of Elevation by Age\nRaw Data only")   
  plt.ylabel(’Elevation␣(m␣IGLD1985)’)   
  plt.xlabel(’Age␣Before␣Present␣(years)’)   
  plt.legend(loc=2, prop={’size’: 17})   
  plt.savefig(’./theDataRaw.png’)   
  plt.close()   
  ############################################################################   
   
   
   
   
  ############################################################################   
  ## create the raw plot with the model included #############################   
  for ds in datasetObjects:   
    print ds, datasetObjects[ds].data   
    x = datasetObjects[ds].getAgeValues()   
    y = datasetObjects[ds].getElevationValues()   
    plt.plot(x, y, mapSiteToColour(ds) + ’s’, label="%s,␣n=%i" % (ds, len(x)), markersize=4.0)   
   
  for d in datasets:   
    plt.plot([age for age in sorted(allAgesSampled) if datasetModels[d].ageValueIsInRangeCoveredByModel(age)], [datasetModels[d].getModelledElevation(age) for age in sorted(allAgesSampled) if datasetModels[d].ageValueIsInRangeCoveredByModel(age)], mapSiteToColour(d), label=d+"␣(model)")   
   
  ##plt.title("Plot of Elevation by Age\nRaw Data with Model")   
  plt.ylabel(’Elevation␣(m␣IGLD1985)’)   
  plt.xlabel(’Age␣Before␣Present␣(years)’)   
  plt.legend(loc=2, prop={’size’: 17})   
  plt.savefig(’./theData.png’)   
  plt.close()   
  ############################################################################   
   
  ############################################################################   
  ## create the raw plot with the model included, zooming in on the 2000-2300#   
  ## ybp window, y axis limited to 183-187 m #################################   
   
   
  zoomXRange = (2000, 2400)   
  zoomYRange = (182, 190)   
   
  for site in datasetObjects:   
    print site, datasetObjects[site].data   
    x = datasetObjects[site].getAgeValues()   
    y = datasetObjects[site].getElevationValues()   
    plt.plot(x, y, mapSiteToColour(site) + ’s’, label="%s" % (site), markersize=4.0)   
   
  for site in datasets:   
    plt.plot(sorted(allAgesSampled), [datasetModels[site].getModelledElevation(age) for age in sorted(allAgesSampled)], mapSiteToColour(site), label=site+"␣(model)")   
    ## plot the dataset models as straight lines   
   
  siteCodeOptions = [site for site in datasetObjects]   
   
  exampleSites = sample(siteCodeOptions, 2)   
  print exampleSites   
   
   
  for order in [’forward’, ’reverse’]:   
   
    if(order == ’forward’):   
      direct = exampleSites[0]   
      modelled = exampleSites[1]   
    elif(order == ’reverse’):   
      direct = exampleSites[1]   
      modelled = exampleSites[0]   
    agesToConsider = [age for age in sorted(allAgesSampled) if ( ((age >= min(zoomXRange))and(age <= max(zoomXRange))) and (datasetModels[direct].ageValueInRawData(age) and datasetModels[modelled].ageValueIsInRangeCoveredByModel(age)) )]   
   
    for age in agesToConsider:   
      print age   
   
    demoComparisonPoint = random.choice(agesToConsider)   
    print "->␣", demoComparisonPoint   
   
   
    directElevation = datasetObjects[direct].getElevationByGivenAge(demoComparisonPoint)   
    modelledElevation = datasetModels[modelled].getModelledElevation(demoComparisonPoint)   
   
    print "Direct␣[%s]:" % direct, directElevation   
    print "Modelled␣[%s]:" % modelled, modelledElevation   
   
    ##plt.plot([demoComparisonPoint, demoComparisonPoint], [directElevation, modelledElevation], "%s" % mapSiteToColour(direct), linewidth=3.0)   
    plt.plot([demoComparisonPoint, demoComparisonPoint], [directElevation, modelledElevation], "%s-." % mapSiteToColour(direct), linewidth=2.0)   
   
  ##’--’   
  ##plt.title("Plot of Elevation by Age\nRaw Data with Model")   
  plt.ylabel(’Elevation␣(m␣IGLD1985)’)   
  plt.xlabel(’Age␣Before␣Present␣(years)’)   
  plt.axis((zoomXRange[0], zoomXRange[1],zoomYRange[0],zoomYRange[1]))   
  plt.legend(loc=2, prop={’size’: 7})   
  plt.savefig(’./theDataZoomed.png’)   
  plt.close()   
  ############################################################################   
   
   
  ############################################################################   
  ## create a list of the shortform name of all the sites, then use ##########   
  ## itertools to make a list of the sites!, all possible combinations of ####   
  ## sites ###################################################################   
   
   
  ## ie [A,B,C] -> [[A,B], [B,C], [C, A]]   
  sites = [ds for ds in datasetObjects]   
  siteCombinations = list(itertools.combinations(sites, 2))   
  ############################################################################   
   
  for combo in siteCombinations:   
    print combo   
    for i in range(len(combo)):   
      print i, combo[i]   
   
   
   
   
   
   
   
   
  globalHistogramFloor = None   
  ## ayy lmao   
   
  histogramFloorsList = []   
   
  histogramFloorsByCombo = {}   
   
  totalAges = []   
   
   
   
  ############################################################################   
  ## loop through the site combinations and use the data to decide on a ######   
  ## floor for the age bins and the bounds on the plot axes ##################   
  for combo in siteCombinations:   
   
    histogramFloor = None   
    ageFloor = None   
    for site in combo:   
      x = datasetObjects[site].getAgeValues()   
      totalAges += x   
      y = datasetObjects[site].getElevationValues()   
      if(histogramFloor == None):   
         histogramFloor = min(y)   
      else:   
         histogramFloor = min([min(y), histogramFloor])   
   
      if(ageFloor == None):   
         ageFloor = min(x)   
      else:   
         ageFloor = min([min(x), ageFloor])   
   
    def roundFloatDownToNearestTen(someFloat):   
      someFloat /= 10   
      someFloat = int(someFloat)   
      someFloat \*= 10   
      someFloat -= 10   
      return someFloat   
   
    histogramFloor = roundFloatDownToNearestTen(histogramFloor)   
    ageFloor = roundFloatDownToNearestTen(ageFloor)   
    histogramFloorsByCombo[combo] = histogramFloor   
   
    histogramFloorsList.append(ageFloor)   
    print "histogramFloor␣for␣site␣combo", combo, ":␣", histogramFloor   
  ############################################################################   
   
   
  globalHistogramFloor = min(histogramFloorsList)   
   
  print "global␣bin␣floor␣set␣at␣", globalHistogramFloor   
   
  globalBins=range(globalHistogramFloor, int(max(totalAges))+200, 200)   
  ## build a list of bin endpoints starting at the floor value and ending at   
  ## one bin width above the last age value of any of the dataset   
   
  ## example of how this works if you run   
  ## range(450, 4857+200, 200)   
  print "global␣bins:␣", globalBins   
   
  ############################################################################   
  ## for debug output print out bin counts for each dataset ##################   
  for i in globalBins:   
    print "bin␣(",i, ",", i+200, "):"   
    print "-"\*80   
    for combo in siteCombinations:   
   
      for site in combo:   
         thisSiteDataset = datasetObjects[site]   
   
         siteName = ’{:4s}’.format(site)   
   
         print "site␣%.4s␣␣count:␣%i" % (siteName, thisSiteDataset.getThisSiteBinCount(i, 200))   
      print "-"\*80   
    print "\n\n"   
  ############################################################################   
   
   
  giaRegressions = {}   
  giaRegressionComboMappingsByConditions = {}   
   
  giaRegressionKeys = []   
  giaRegressionDescriptions = {}   
  giaKeysByDescriptions = {}   
   
  for combo in siteCombinations:   
    for order in [’forward’, ’reverse’]:   
      if(order == ’forward’):   
         direct = combo[0]   
         ## d is the site we are using as our direct comparison   
   
         ## ie MUST have a measured data point at this age   
         modelled = combo[1]   
         ## ds is what we are comparing against, so it can just be a   
         ## modelled point   
      else:   
         direct = combo[1]   
         modelled = combo[0]   
      thisRegressionKey = "%s-%s:%s" % (combo[0], combo[1], order)   
   
      giaRegressionKeys.append(thisRegressionKey)   
      thisComparisonGiaDescription = "%s␣relative␣to␣%s␣model" % (direct, modelled)   
      giaRegressionDescriptions[thisRegressionKey] = thisComparisonGiaDescription   
      giaKeysByDescriptions[thisComparisonGiaDescription] = thisRegressionKey   
   
  sortedKeys = sorted(giaRegressionKeys)   
  print "sortedKeys:␣", sortedKeys   
   
  ############################################################################   
  ## plot the raw data plots with counts for each bin ########################   
   
  for combo in siteCombinations:   
    print "\nPlotting␣raw␣data␣for␣site␣combo:␣"   
    for site in combo:   
      print site   
   
    for site in combo:   
      x = datasetObjects[site].getAgeValues()   
      y = datasetObjects[site].getElevationValues()   
   
   
      plt.plot(x, y, mapSiteToColour(site) + ’s’, label="%s,␣n=%i" % (site, len(x)), markersize=4.0)   
      ## plot the raw data for each site   
   
      plt.plot(sorted(allAgesSampled), [datasetModels[site].getModelledElevation(age) for age in sorted(allAgesSampled)], mapSiteToColour(site), label=site+"␣(model)")   
      ## plot the linear interpolation model for each site   
   
      plt.hist(x, bottom = histogramFloor, normed=False, bins=globalBins, alpha=0.4, color=mapSiteToColour(site))   
      ## plot the histogram of data set counts on the plot alongside the   
      ## data itself   
   
      ## histogram floor was chosen here as a nice looking spot to put the   
      ## count histogram so it doesnt overlap the main data   
   
   
    ##plt.title("Data and Model for site Combination %s/%s" % (combo[0], combo[1]))   
    plt.ylabel(’Elevation␣(m␣IGLD1985)’)   
    plt.xlabel(’Age␣Before␣Present␣(Years)’)   
    plt.legend(loc=2, prop={’size’: 17})   
   
   
    axes1 = plt.gca()   
    yScaleRange = max(axes1.get\_ylim()) - min(axes1.get\_ylim())   
   
   
    axes2 = plt.twinx()   
    axes2.set\_ylabel(’Count’)   
    axes2.axis([None,None,0,yScaleRange])   
    ## set the left axis to be elevation relative to datum   
   
    ## and the right axis to be count of each dataset in each bin   
   
    outputFilePath = filePathOnRelativePath("./", fileName=’%s-%s\_DataAndModel’ % (combo[0], combo[1]), ext="png")   
    print "Saving␣rawData␣plot␣at␣’%s’" % outputFilePath   
    verifyPath("./")   
    ## umm ok   
    plt.savefig(outputFilePath)   
    plt.close()   
    ## save the raw data combo graph   
  ############################################################################   
   
   
  ############################################################################   
  ## plot the gia graphs and store the raw regression numbers used to create #   
  ## them ####################################################################   
  for conditions in [{"valueCounts": "bothNonZero"},\   
             ##{"valueDifference": "withinThirtyPercent","valueCounts": "bothNonZero"},\   
             ##{"valueDifference": "withinFiftyPercent","valueCounts": "bothNonZero"}, \   
             ##{"valueDifference": "withinTwentyPercent","valueCounts": "bothNonZero"}, \   
             {"valueDifference": "withinSeventyFivePercent","valueCounts": "bothNonZero"}]:   
    outputPathDict = populateConditionsDict(conditions)   
   
    outputPath = convertListToRelativePath([outputPathDict[setting] for setting in getCurrentSettingOptions()])   
    conditionIdString = "\_".join([outputPathDict[setting] for setting in getCurrentSettingOptions()])   
   
    giaRegressionsByCombo = {}   
   
    for combo in siteCombinations:   
   
   
      print "\nPlotting␣gia␣for␣site␣combo:␣"   
      for site in combo:   
         print site   
   
   
   
      ## now the gia calculations   
      for order in [’forward’, ’reverse’]:   
         ## each comparison has a forward A to B, and reverse B to A,   
         ## comparison, the CIs on the absolute value of slope for this must   
         ## be statistically similar for the comparison to work   
   
         if(order == ’forward’):   
           direct = combo[0]   
           ## d is the site we are using as our direct comparison   
   
           ## ie MUST have a measured data point at this age   
           modelled = combo[1]   
           ## ds is what we are comparing against, so it can just be a   
           ## modelled point   
         else:   
           direct = combo[1]   
           modelled = combo[0]   
   
   
   
         allowableAgeValues = []   
         ## for each comparison, there are only a small number of data values   
         ## from the initial dataset that can be used for valid comparison   
   
         ## each datapoint used for a gia comparison must be:   
         ## -from the direct dataset   
         ## -in the range covered by the modelled dataset (meaning that if   
         ## the direct comparison dataset has a datapoint available, but the   
         ## modelled one has just been hanging off the end in a straight line   
         ## from the last known datapoint, it cant be considered valid   
         ## -given that theres a bin from startAge to startAge+binWidth that   
         ## the datapoints age is in, that bin needs to hit some criteria for   
         ## the number of datapoints in the bin from both   
   
         for age in sorted(allAgesSampled):   
           if(datasetModels[direct].ageValueInRawData(age) and datasetModels[modelled].ageValueIsInRangeCoveredByModel(age) and datasetModels[modelled].ageComparisonValidForThisBin(datasetModels[direct], globalBins, age, conditions) ):   
             allowableAgeValues.append(age)   
           else:   
             continue   
             ## the case where we have an overlap of the models, but   
             ## either A: no datapoint is actually present for either   
             ## dataset at this age, so comparisons are not honouring   
             ## the raw data, or   
             ## B: we have a datapoint on the set to compare against   
             ## but not the one we are comparing   
             ##else:   
               ##continue   
               ## if the datapoint in question is outside the bounds   
               ## covered by these two datasets, they cant be considered   
   
   
         elevationDiffs = [(datasetModels[direct].getModelledElevation(age) - datasetModels[modelled].getModelledElevation(age)) for age in allowableAgeValues]   
   
   
         bootstrapper.plotBootstrapsOnDataPlot(plt, allowableAgeValues, elevationDiffs, mapSiteToColour(modelled), mapSiteToColour(direct));   
   
         thisComparisonGiaDescription = "␣measured␣data␣from␣%s␣relative␣to␣%s␣model" % (direct, modelled)   
         plt.plot(allowableAgeValues, elevationDiffs, mapSiteToColour(direct)+’+’, label=thisComparisonGiaDescription, markersize=4.0)   
   
   
         linRegressYValues, gradient, intercept, gradientError, yModelHigh, yModelLow, rSquare = getLinearModel(allowableAgeValues, elevationDiffs)   
   
         if(direct != modelled):   
           giaRegressionKey = "%s-%s:%s" % (combo[0], combo[1], order)   
   
           giaRegressionsByCombo[giaRegressionKey] = {"N": len(allowableAgeValues), "gradientEstimator": gradient, "gradientError": gradientError, "gradient": [gradient+(1.96\*gradientError), gradient-(1.96\*gradientError)], "intercept": intercept, "rSquare": rSquare}   
   
         plt.suptitle("Plot␣of␣Elevation␣Diff␣for␣%s␣relative␣to␣%s␣model␣by␣Age\n(%s␣order␣for␣%s-%s),␣n=%i\ny=mx+b,␣m␣=␣%.4f␣SE(%.4f),␣b␣=␣%.4f,␣r^2␣=␣%.3f" % (direct, modelled, order, combo[0], combo[1], len(allowableAgeValues), gradient, gradientError, intercept, rSquare), fontsize=10)   
         plt.ylabel(’Elevation’)   
         plt.xlabel(’Age’)   
         if(direct == "ATB"):   
           plt.legend(loc=2, prop={’size’: 14})   
   
         else:   
           plt.legend( loc=3, prop={’size’: 14})   
         ##plt.savefig(’./theGIA\_%s\_relative\_to\_%s.png’ % (d, ds))   
         ## ^ this was creating a ton of clutter   
   
   
         outputFilePath = filePathOnRelativePath(outputPath+"gias/", fileName=’theGIA\_%s\_relative\_to\_%s’ % (direct, modelled), ext="png")   
         print "Saving␣gia␣plot␣at␣’%s’" % outputFilePath   
         verifyPath(outputPath+"gias/")   
         plt.savefig(outputFilePath)   
         plt.close()   
    giaRegressionComboMappingsByConditions[conditionIdString] = giaRegressionsByCombo   
    plotGradientConfidenceIntervals(giaRegressionsByCombo, giaRegressionKeys, giaRegressionDescriptions, outputPathDict)   
  ############################################################################   
  print "Finished␣gia␣plots"   
   
   
  ############################################################################   
  ## Check for any exact age matches in the datasets provided ################   
  ## Spoiler: there arent any ################################################   
  ageMatches = []   
  for d in datasets:   
    for dv in datasetObjects[d].getAgeValues():   
      for od in datasets:   
         if(od != d):   
           if((dv in datasetObjects[od].getAgeValues())and dv not in ageMatches):   
             ageMatches.append(dv)   
  print "Exact␣age␣matches␣between␣datasets:␣", ageMatches   
  ############################################################################   
   
   
   
   
  ############################################################################   
  ## now that values have been generated for GIA for each site comparison, ###   
  ## convert them to intervals for each site combination and save the result #   
  ## to file #################################################################   
  for idString in giaRegressionComboMappingsByConditions:   
    print "\n\n%s:␣" % idString   
   
    giaRegressionsByCombo = giaRegressionComboMappingsByConditions[idString]   
    siteCombos = ["ATB-BATB","GTB-ATB","GTB-BATB","GTB-TAHB","TAHB-ATB","TAHB-BATB"]   
   
    with open("%s\_intervals.csv" % idString, "wb") as csv\_file:   
      writer = csv.writer(csv\_file, delimiter=’,’)   
   
      writer.writerow([ "name", "Slope␣Estimator", "Slope␣Error", "r␣Squared", "Slope␣C.I.␣(95p)"])   
      for regress in sortedKeys:   
         description = giaRegressionDescriptions[regress]   
         ciStart = 100\*100\*giaRegressionsByCombo[regress][’gradient’][0]   
         ciEnd = 100\*100\*giaRegressionsByCombo[regress][’gradient’][1]   
         est = 100\*100\*giaRegressionsByCombo[regress][’gradientEstimator’]   
         error = 100\*100\*giaRegressionsByCombo[regress][’gradientError’]   
         rSquare = giaRegressionsByCombo[regress][’rSquare’]   
   
         print regress   
         for param in giaRegressionsByCombo[regress]:   
           print param   
   
         writer.writerow([ description, "%.5f" % est, "%.5f" % error, "%.3f" % rSquare, "%.5f␣to␣%.5f" % (ciStart, ciEnd)])   
   
    for combo in siteCombos:   
      comboSites = combo.split(’-’)   
   
      print combo, comboSites   
   
   
      with open("%s\_regressionTable\_%s.csv" % (combo, idString), "wb") as csv\_file:   
         writer = csv.writer(csv\_file, delimiter=’,’)   
         writer.writerow([ "name", "Slope␣Estimator", "Slope␣Error", "r␣Squared", "Slope␣C.I.␣(95p)"])   
         for order in ["forward", "reverse"]:   
           regress = "%s:%s" % (combo, order)   
           description = giaRegressionDescriptions[regress]   
           ciStart = 100\*100\*giaRegressionsByCombo[regress][’gradient’][0]   
           ciEnd = 100\*100\*giaRegressionsByCombo[regress][’gradient’][1]   
           est = 100\*100\*giaRegressionsByCombo[regress][’gradientEstimator’]   
           error = 100\*100\*giaRegressionsByCombo[regress][’gradientError’]   
           rSquare = giaRegressionsByCombo[regress][’rSquare’]   
           writer.writerow([ description, "%.5f" % est, "%.5f" % error, "%.3f" % rSquare, "%.5f␣to␣%.5f" % (ciStart, ciEnd)])   
   
   
    with open("%s\_mergedIntervals.csv" % idString, "wb") as csv\_file:   
      writer = csv.writer(csv\_file, delimiter=’,’)   
   
      writer.writerow(["siteCombination", "startValue", "endValue"])   
      for combo in siteCombos:   
         for order in ["forward", "reverse"]:   
           regress = "%s:%s" % (combo, order)   
           print giaRegressionDescriptions[regress], ",", giaRegressionsByCombo[regress][’gradient’][0], ",", giaRegressionsByCombo[regress][’gradient’][1]   
   
           est = giaRegressionsByCombo[regress][’gradientEstimator’]   
           ciStart = 100\*100\*giaRegressionsByCombo[regress][’gradient’][0]   
           ciEnd = 100\*100\*giaRegressionsByCombo[regress][’gradient’][1]   
   
           if(est < 0):   
             ciStart = -ciStart   
             ciEnd = -ciEnd   
   
           if(order == "forward"):   
             forwardInterval = {"start":min(ciStart, ciEnd), "end": max(ciStart, ciEnd)}   
           elif(order == "reverse"):   
             reverseInterval = {"start":min(ciStart, ciEnd), "end": max(ciStart, ciEnd)}   
   
         mergedInterval = mergeConfidenceIntervals(forwardInterval, reverseInterval)   
         print combo, "␣", "␣forward:␣", forwardInterval, "␣reverse:␣", reverseInterval, "␣merged:␣", mergedInterval   
         if(mergedInterval == "No␣overlap"):   
           writer.writerow([combo, "%s\_merged" % combo, mergedInterval, ""])   
         else:   
           writer.writerow([combo, "%s\_merged" % combo, "%.3f" % mergedInterval[0], "%.3f" % mergedInterval[1]])   
   
   
   
  ############################################################################   
  ## plot a legend showing the colour coding system for the sites ############   
  ## this sounded like a decent idea earlier, but it eventually proved #######   
  ## not to be needed ########################################################   
  for site in sites:   
    plt.plot([1], [1], mapSiteToColour(site)+’s’, label=site, markersize=20)   
    plt.plot([1], [1], mapSiteToColour(site), label=site+"␣model", markersize=20)   
  plt.axis(’off’)   
  plt.legend(loc=3, prop={’size’: 29})   
  plt.savefig("legendary.png")   
  plt.close()   
  ############################################################################

7.2 Source code for giaUtils.py

   
## giaUtils.py #################################################################   
## utility functions for gia thesis ############################################b   
################################################################################   
import os   
   
   
def mapSiteToColour(siteLoc):   
  siteMappings ={’BATB’: ’b’, ’TAHB’: ’g’, ’GTB’: ’m’, ’ATB’: ’r’}   
  return siteMappings[siteLoc]   
   
def convertListToRelativePath(someListOfStrings):   
  output = "./"   
  for someStr in someListOfStrings:   
    output += "%s/" % someStr   
  return output   
   
def filePathOnRelativePath(somePath, fileName, ext=None):   
  if(ext == None):   
    return "%s%s" % (somePath, fileName)   
  else:   
    return "%s%s.%s" % (somePath, fileName, ext)   
   
def verifyPath(somePath):   
  if(os.path.exists(somePath)):   
    if(os.path.isdir(somePath)):   
      return True   
    else:   
      print "Path␣’%s’␣exists,␣but␣is␣not␣a␣directory"   
      return False   
  else:   
    print "Path␣did␣not␣exist,␣attempting␣to␣create␣it..."   
    os.makedirs(somePath)   
    return os.path.exists(somePath)   
   
   
## input dict   
## ie {"valueDifference": "withinTwentyPercent","valueCounts": "bothNonZero"}   
   
def populateConditionsDict(inputDict):   
  if("valueDifference" not in inputDict):   
    inputDict["valueDifference"] = "any"   
  if("valueCounts" not in inputDict):   
    inputDict["valueCounts"] = "any"   
  return inputDict   
  ## technically it mutates the dict, but this is ok too   
   
def getCurrentSettingOptions():   
  return [ "valueCounts", "valueDifference"]   
   
   
   
def mergeConfidenceIntervals(intervalA, intervalB):   
   
  if(intervalA["start"] > intervalA["end"]):   
    realStart = intervalA["end"]   
    realEnd = intervalA["start"]   
    intervalA = {"start": realStart, "end": realEnd}   
   
  if(intervalB["start"] > intervalB["end"]):   
    realStart = intervalB["end"]   
    realEnd = intervalB["start"]   
    intervalB = {"start": realStart, "end": realEnd}   
   
  if((intervalA["start"] >= intervalB["end"]) or (intervalB["start"] >= intervalA["end"])):   
    return "No␣overlap"   
  else:   
    ## some overlap   
    if((intervalB["start"] < intervalA["start"]) and (intervalB["end"] < intervalA["end"])):   
      ##print "case 1"   
      return (intervalA["start"], intervalB["end"])   
    elif((intervalA["start"] < intervalB["start"]) and (intervalA["end"] < intervalB["end"])):   
      ##print "case 2"   
      return (intervalB["start"], intervalA["end"])   
   
    elif((intervalB["start"] > intervalA["start"]) and (intervalB["end"] < intervalA["end"])):   
      ##print "B contained case"   
      return (intervalB["start"], intervalB["end"])   
    elif((intervalA["start"] > intervalB["start"]) and (intervalA["end"] < intervalB["end"])):   
      ##print "A contained case"   
      return (intervalA["start"], intervalA["end"])   
   
   
if(\_\_name\_\_ == "\_\_main\_\_"):   
  print convertListToRelativePath(["withinTwentyPercent", "baseFixedAt450", "gias"])   
   
  forward = {’start’: 5.7478045853453503, ’end’: 13.427190249363679}   
  reverse = {’start’: 8.9698781911037724, ’end’: 16.578153894106137}   
  print mergeConfidenceIntervals(forward, reverse)   
  print mergeConfidenceIntervals(reverse, forward)

7.3 Source code for rawhide/bootstrapper.py

   
## bootstrapper.py ############################################################   
## tools to create a bootstrap for a linear regression model, and plot this ##   
## in matplotlib ###############################################################   
###############################################################################   
   
import numpy as np   
import matplotlib.pyplot as plt   
from sklearn.linear\_model import LinearRegression   
   
def plotBootstrapsOnDataPlot(pllt, x, y, strapColor=’grey’, regressColor=’red’):   
   
   
   
  # Extend x data to contain another row vector of 1s   
  x = np.asarray(x)   
  y = np.asarray(y)   
   
  X = np.vstack([x, np.ones(len(x))]).T   
   
  plt.figure(figsize=(12,8))   
  for i in range(0, 500):   
    sample\_index = np.random.choice(range(0, len(y)), len(y))   
   
    X\_samples = X[sample\_index]   
    y\_samples = y[sample\_index]   
   
    lr = LinearRegression()   
    lr.fit(X\_samples, y\_samples)   
    plt.plot(x, lr.predict(X), color=strapColor, alpha=0.1, zorder=1)   
   
  lr = LinearRegression()   
  lr.fit(X, y)   
  plt.plot(x, lr.predict(X), color=regressColor, zorder=5)   
   
if(\_\_name\_\_ == "\_\_main\_\_"):   
  ## Create toy data and bootstrap it to verify everything is working   
  ## correctly   
   
  x = np.linspace(0, 10, 20)   
  y = x + (np.random.rand(len(x)) \* 10)   
  plotBootstrapsOnDataPlot(plt, x, y)   
  plt.scatter(x, y, marker=’+’, color=’blue’, zorder=5)   
   
  plt.savefig(’boostrapDemo.png’)

7.4 Source code for rawData.py

   
## rawData.py ##################################################################   
## object containing the raw data object used to wrap what gets pulled out #####   
## of the spreadsheet file #####################################################   
################################################################################   
   
   
   
class siteData(object):   
   
  ## siteData: Str Listof(Any) -> siteData   
  def \_\_init\_\_(self, siteName, rawData):   
    self.dataHeader = []   
    self.data = [row for row in rawData if len(row) == 2]   
    ## filter to only those rows that contain exactly two elements in order   
    ## to get rid of the unimportant details at the top of each sheet   
    self.siteName = siteName   
   
   
  def getAgeValues(self):   
    ## get all available (measured) data points for age   
    return [row[1] for row in self.data]   
   
   
  def getElevationValues(self):   
    ## get the entire list of elevation values for this dataset   
    return [row[0] for row in self.data]   
   
  def getElevationByGivenAge(self, someAge):   
    ## map an age to an elevation if possible   
    for row in self.data:   
      if(row[1] == someAge):   
         return row[0]   
   
  def getThisSiteBinCount(self, binStart, binWidth):   
   
    def withinside(someValue, binStart, binWidth):   
      ## determine if someValue is between binStart and (binStart+binWidth)   
      delta = someValue - binStart   
      if((delta >= 0)and(delta <= binWidth)):   
         return True   
      else:   
         return False   
   
    return len([row[1] for row in self.data if withinside(row[1], binStart, binWidth)])   
   
  def getSiteName(self):   
    return self.siteName

7.5 Source code for dataModel.py

   
## dataModel.py ################################################################   
## Base class for building data models (ie interpretations of what the age #####   
## is for the entire age range that the data spans) ############################   
################################################################################   
from rawData import \*   
   
   
## ideas for future models:   
## -same connect the dots idea, but with binned means every so many years   
## -use mean water level of other sites in areas where a site has a gap to   
## adjust for global water lows caused by climate, etc.   
   
   
class siteModel(object):   
  ## parent to all models that take a set of siteData and attempt to build a   
  ## model of elevations for all of the possible age values in between points   
  ## where the elevation is directly sampled for that exact time.   
   
   
   
  def \_\_init\_\_(self):   
    pass   
   
  ## getModelledElevation: Num -> Num   
   
   
   
   
def inRange(value, high, low):   
  if((value <= high) and (value >= low)):   
    return True   
  return False   
   
   
## getAgeBinByAgeValue: float, listof(float) -> float, float   
   
## ageValue is a float   
## ageBins is some list of bin endpoints   
   
def getAgeBinByAgeValue(ageValue, ageBins):   
  baseAge = ageBins[0]   
  ageBinsDelta = ageBins[1] - baseAge   
  ## should be consistent throughout the ageBins list   
   
  ## now find the nearest start point for a bin to the ageValue   
   
  binStartValue = baseAge   
   
  if(ageValue < baseAge):   
    while(True):   
      if(inRange(ageValue, binStartValue+ageBinsDelta, binStartValue)):   
         return binStartValue, ageBinsDelta   
      binStartValue -= ageBinsDelta   
  else:   
    while(True):   
      if(inRange(ageValue, binStartValue+ageBinsDelta, binStartValue)):   
         return binStartValue, ageBinsDelta   
      binStartValue += ageBinsDelta

7.6 Source code for linearInterpolationModel.py

   
## linearInterpolationModel.py #################################################   
## model for elevation v. time values made by connecting point to point, ie ####   
## "connect the dots" formally known as linear interpolation ###################   
################################################################################   
from dataModel import \*   
   
import numpy as np   
   
   
def percentageDifference(someValue, anotherValue):   
  average = float(someValue + anotherValue)/2.0   
  diff = abs(someValue - anotherValue)   
  return float(diff/average)   
   
   
def conditionMet(thisBinCount, otherBinCount, condition):   
  if(condition == "any"):   
    return True   
  elif(condition == "bothNonZero"):   
    if((thisBinCount > 0) and (otherBinCount > 0)):   
      return True   
    return False   
  elif(condition == "withinTwentyPercent"):   
    if(percentageDifference(thisBinCount, otherBinCount) <= 0.20):   
      return True   
    return False   
  elif(condition == "withinThirtyPercent"):   
    if(percentageDifference(thisBinCount, otherBinCount) <= 0.30):   
      return True   
    return False   
  elif(condition == "withinFiftyPercent"):   
    if(percentageDifference(thisBinCount, otherBinCount) <= 0.50):   
      return True   
    return False   
  elif(condition == "withinSixtyPercent"):   
    if(percentageDifference(thisBinCount, otherBinCount) <= 0.60):   
      return True   
    return False   
  elif(condition == "withinSeventyFivePercent"):   
    if(percentageDifference(thisBinCount, otherBinCount) <= 0.75):   
      return True   
    return False   
   
   
class siteModelConnectTheDots(siteModel):   
   
  ## siteModelConnectTheDots: siteData -> siteModelConnectTheDots   
   
  def \_\_init\_\_(self, availableData):   
    self.siteName = availableData.getSiteName()   
    self.rawDataObject = availableData   
   
  def ageValueIsInRangeCoveredByModel(self, someAge):   
    if(someAge in self.rawDataObject.getAgeValues()):   
      return True   
    else:   
      maxAgeCovered = max(self.rawDataObject.getAgeValues())   
      minAgeCovered = min(self.rawDataObject.getAgeValues())   
      if((someAge <= maxAgeCovered)and(someAge >= minAgeCovered)):   
         return True   
      else:   
         return False   
   
  def ageValueInRawData(self, someAge):   
    if(someAge in self.rawDataObject.getAgeValues()):   
      return True   
    return False   
   
  def ageComparisonValidForThisBin(self, otherModelToCompareAgainst, globalBins, ageValue, conditions):   
    binStart, binWidth = getAgeBinByAgeValue(ageValue, globalBins)   
   
    thisModelsBinCount = self.rawDataObject.getThisSiteBinCount(binStart, binWidth)   
    otherModelsBinCount = otherModelToCompareAgainst.rawDataObject.getThisSiteBinCount(binStart, binWidth)   
   
    for condition in conditions:   
      if(not conditionMet(thisModelsBinCount, otherModelsBinCount, conditions[condition])):   
         return False   
    return True   
    ## the loop successfully met every condition, so we’re good to go   
   
   
  def getModelledElevation(self, someAge):   
    if(someAge in self.rawDataObject.getAgeValues()):   
      return self.rawDataObject.getElevationByGivenAge(someAge)   
    else:   
      ## dont have a datapoint available at that age value, so we need to   
      ## interpolate linearly between them to get it   
      ageValues = np.array(self.rawDataObject.getAgeValues())   
   
      if(ageValues[ageValues < someAge].size == 0):   
         ## case where our value to interpolate is off the bottom end   
         ## of the dataset, so we extrapolate from the last two values   
         ## min, and 2dmin   
   
         minValue = min(ageValues)   
         restOfValues = np.array([val for val in ageValues if val != minValue])   
         ## maybe npifying the array will make the min/max calls faster   
         ## idk   
         secondMinValue = min(restOfValues)   
         ageDelta = someAge - secondMinValue   
         ## distance from second smallest to the point we want to   
         ## interpolate   
   
   
   
         secondMinAgeElevation = self.rawDataObject.getElevationByGivenAge(secondMinValue)   
         minAgeElevation = self.rawDataObject.getElevationByGivenAge(minValue)   
   
         outputElevationGuess = secondMinAgeElevation + ( (ageDelta/(abs(secondMinValue-minValue)))\*(secondMinAgeElevation - minAgeElevation) )   
   
      elif(ageValues[ageValues > someAge].size == 0):   
         ## case where our value to interpolate is off the top end   
         maxValue = max(ageValues)   
         restOfValues = np.array([val for val in ageValues if val != maxValue])   
         ## npifying this array could make the min/max calls faster   
   
         secondMaxValue = max(restOfValues)   
   
         ageDelta = someAge - secondMaxValue   
   
         secondMaxAgeElevation = self.rawDataObject.getElevationByGivenAge(secondMaxValue)   
         maxAgeElevation = self.rawDataObject.getElevationByGivenAge(maxValue)   
   
         outputElevationGuess = secondMaxAgeElevation + ( (ageDelta/(abs(maxValue - secondMaxValue)))\*(maxAgeElevation - secondMaxAgeElevation) )   
   
      else:   
         closestAgeBelow = ageValues[ageValues < someAge].max()   
         closestAgeAbove = ageValues[ageValues > someAge].min()   
   
   
   
   
         ageDelta = closestAgeAbove - closestAgeBelow   
   
         elevBelow = self.rawDataObject.getElevationByGivenAge(closestAgeBelow)   
         elevAbove = self.rawDataObject.getElevationByGivenAge(closestAgeAbove)   
   
         elevDelta = elevAbove - elevBelow   
   
         outputElevationGuess = elevBelow + elevDelta\*((someAge - closestAgeBelow)/(ageDelta))   
   
      return outputElevationGuess