Preprint typeset using LaTeX style emulateapj v. 11/10/09

CHOPSTACKS

Hajime Kawahara[1,2]

ABSTRACT

ChopStacks is a python code collection for resampling data and stacking them under the flux preservation.

1. INTRODUCTION

We sometimes need to re-sample 1D discrete data (i.e. spectra, light curves, and so on). Of course, the interpolation is one of the major solutions to this problem. However, we sometimes need the flux preservation simultaneously, for instance, when we stack the data.My Python small package, Chopstacks, provides the functions,

- re-sampling 1D data and preserving flux with an arbitrary array,
- stacking 1D data with preserving flux,
- converting the 1D data into the log 1D data with preserving flux.

This document describes the background mathematics of Chopstacks. If you want to know how to use, read jupyter notebooks in ipynb directory.

2. ALGORITHM

I consider the situation that the stacked spectrum (we use the term of "spectrum" as an example of 1D data) $\hat{f}(x)$ consists of the sum of M-spectra as

$$\hat{f}(x) = \sum_{m=0}^{M-1} f^{(m)}(x) \tag{1}$$

I assume that the stacked spectrum and each spectrum have the unit of $[q/X]$, where q is the unit of preserved quantity $Q(x)$ and X is the unit of x.

The stacked spectrum $\hat{f}$ is discretized to the master bin as

$$\hat{f}_i \equiv \hat{f}(\hat{x}_i) \tag{2}$$
$$\hat{x}_i \ : \ i = 0, 1.., N-1. \tag{3}$$
$$f_j^{(m)} \equiv f^{(m)}(x_j) \tag{4}$$
$$x_j \ : \ j = 0, 1.., N-1. \tag{5}$$

The width of each bin $\hat{x}_i$ is defined by $\hat{x}_{i,s}$ and $\hat{x}_{i,e}$ as

$$\Delta\hat{x}_i \equiv \hat{x}_{i,e} - \hat{x}_{i,s}. \tag{6}$$

The quantity in the i-th bin is written as

$$\hat{Q}_i = \hat{f}_i \Delta\hat{x}_i \tag{7}$$

and

$$Q_j^{(m)} = f_j^{(m)} \Delta x_j^{(m)}. \tag{8}$$

kawahara@eps.s.u-tokyo.ac.jp
[1] Department of Earth and Planetary Science, The University of Tokyo, Tokyo 113-0033, Japan
[2] Research Center for the Early Universe, School of Science, The University of Tokyo, Tokyo 113-0033, Japan

The problem is how to distribute the quantity in $\hat{f}_i$ from $f_j^{(m)}$. I denote the overlaped region between $\hat{x}_i$ and $x_j^{(m)}$ by $\Delta_{i,j}^{(m)}$. Then the contribution of the j-th bin of the m-th spectra to $\hat{Q}_i$ is

$$\Delta Q_{i;j}^{(m)} = Q_j^{(m)} \frac{\Delta_{i,j}^{(m)}}{\Delta x_j^{(m)}} \tag{9}$$
$$= f_j^{(m)} \Delta_{i,j}^{(m)} \tag{10}$$

Hence, the quantity in the i-th stacked bin is

$$\hat{Q}_i = \sum_{m=0}^{M-1} \sum_j \Delta Q_{i;j}^{(m)} \tag{11}$$
$$= \sum_{m=0}^{M-1} \sum_j f_j^{(m)} \Delta_{i,j}^{(m)}. \tag{12}$$

Using equation (7), I obtain

$$\hat{f}_i = \sum_{m=0}^{M-1} \sum_{j=0}^{N-1} f_j^{(m)} \frac{\Delta_{i,j}^{(m)}}{\Delta\hat{x}_i}. \tag{13}$$

Though this eqution has $M \times N$ terms, however, most of $\Delta_{i,j}^{(m)}$ is zero. Hence, it is faster if I can find the overlap (i, j) which satisfies $\Delta_{i,j}^{(m)} \neq 0$ for each spectrum.

$$\hat{f}_i = \sum_{m=0}^{M-1} \sum_{j \in \mathrm{overlap}} f_j^{(m)} \frac{\Delta_{i,j}^{(m)}}{\Delta\hat{x}_i}. \tag{14}$$

2.1. *implementation*

chopstacks.py is the central program of this algorithm. The main subroutine to redistribute the value is **cutput** in **chopstacks.py**. The illustrative explanation of this routine is shown in Figure 1. The input arrays of **cutput** are

- xw: walls of the original bins [X], i.e. (N)-array.
- f: input value [q/X], i.e. (N-1)-array.
- hxw: walls of the resampled bins [X], i.e. (N)-array.
- hf: stacking array [q/X], i.e. (N-1)-array,

and the output is

- hf: the input hf + resampled and stacked data.

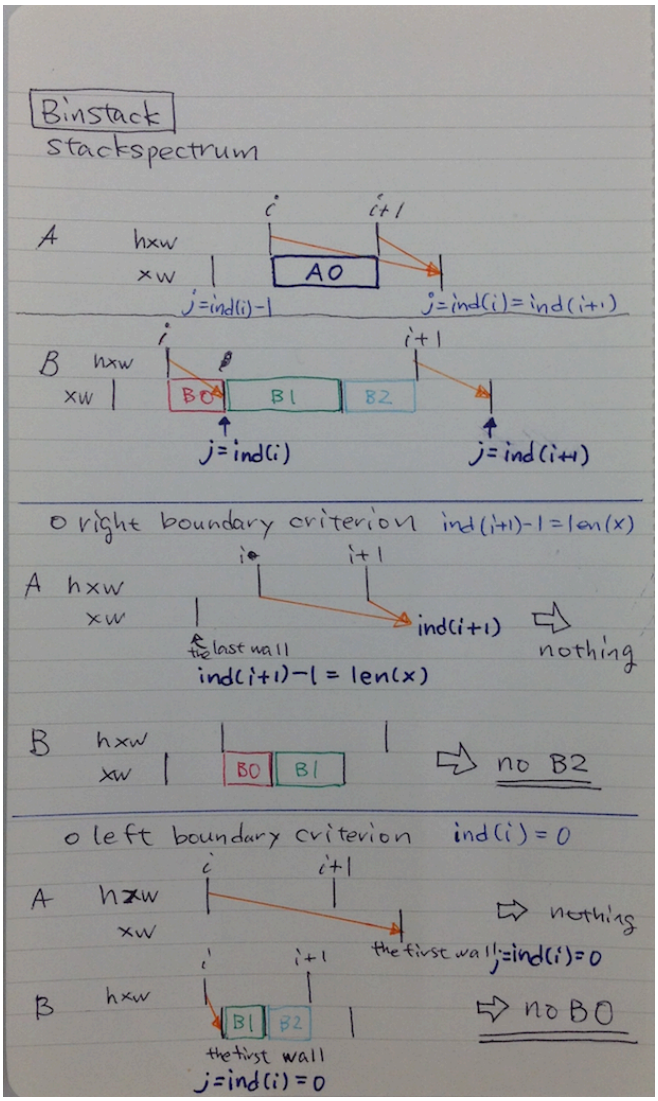

Figure 1. Schematic explanation of **cutput**. The code is shown in appendix.

Because 1D data often use the representative values of [X] (usually, the central value of the bin), you need to set N-walls between the $(N-1)$-representative values of [X]. If you want to set the walls to be the center of the neighboring representative values, use **buildwall** in chopstacks.py. The edge option specifies the deal of the edge values. If you specify hf=None, then chopstacks.py assumes the initial hf to be a zero array. So, if you just want to divide the 1D data, use hf=None.

3. APPLICATIONS

Use the spectra with the unit of $[q/X]$. The preservation of q can be checked using **check_preservation**.

3.1. *Log Bin Convertor*

analogbin.py perform the resampling of f $[q/X]$ into the logscale of x ($l \equiv \log x$) with a resolution power R. The relation between $\delta \log x$ and R is provided by

$$\Delta l \equiv \delta \log x = \frac{\Delta x}{x} = \frac{1}{R}. \tag{15}$$

Then the number of the rebinned data is given by

$$\hat{N} = \frac{\log \hat{x}_N - \log \hat{x}_0}{\Delta l} = R(\log \hat{x}_N - \log \hat{x}_0). \tag{16}$$

The **opt** option specifies the output form. opt=0 will provide

- $\hat{x}$, walls of $\hat{x}$, $\hat{f}$.

opt=1 provides

- $\log \hat{x}$, walls of $\log \hat{x}$, $\hat{x}\hat{f}$.

The latter output is based on the preservation law

$$\begin{aligned} f dx &= f \left| \frac{\partial x}{\partial (\log x)} \right| d(\log) x \\ &= x f d(\log x). \end{aligned} \tag{17}$$

Subroutine **check_preservation** can check the degree of preservation of $[q]$.

APPENDIX

```
def cutput(xw,f,hxw,hf=None,silent=None):
    #see note 14.8.12 and 14.9.02
    if hf is None:
        if silent is None:
            print "Reset master data (hf) to zero."
        hf=np.zeros(len(hx))

    ind=np.digitize(hxw,xw)
    for i in range(0,len(hxw)-1):

        if ind[i]<len(xw) and ind[i+1]-1 < len(xw)+1 and ind[i]>0:  #cx1,cx2,cx3
            if ind[i]==ind[i+1]:
                hf[i]=hf[i]+f[ind[i]-1] #A0
            else:
                hf[i]=hf[i]+f[ind[i]-1]*(xw[ind[i]]-hxw[i])/(hxw[i+1]-hxw[i]) #B0
                for k in range(ind[i]+1, ind[i+1]):
                    hf[i]=hf[i]+f[k-1]*(xw[k]-xw[k-1])/(hxw[i+1]-hxw[i]) #B1
                hf[i]=hf[i]+f[ind[i+1]-1]*(hxw[i+1]-xw[ind[i+1]-1])/(hxw[i+1]-hxw[i]) #B2

        elif ind[i+1]-1 == len(xw)+1:  #right boundary criterion
            if ind[i]<ind[i+1]:
                hf[i]=hf[i]+f[ind[i]-1]*(xw[ind[i]]-hxw[i])/(hxw[i+1]-hxw[i]) #B0
                for k in range(ind[i]+1, ind[i+1]):
                    hf[i]=hf[i]+f[k-1]*(xw[k]-xw[k-1])/(hxw[i+1]-hxw[i]) #B1

        elif ind[i]==0 and ind[i+1]>0: #left boundary condition
                for k in range(ind[i]+1, ind[i+1]):
                    hf[i]=hf[i]+f[k-1]*(xw[k]-xw[k-1])/(hxw[i+1]-hxw[i]) #B1
                hf[i]=hf[i]+f[ind[i+1]-1]*(hxw[i+1]-xw[ind[i+1]-1])/(hxw[i+1]-hxw[i]) #B2

    return hf
```