HeatMap Manual

Andrew Dotson

August 2, 2018

Abstract

This document should serve as a reference for those wanting to run or make any changes to the code "PublicR2R3.ipynb".

1 Description

"PublicR2R3.ipynb" was written to take in specific values of kinematic variables and momentum fractions, and return a heatmaps of Regions 2 and 3, which correspond to the transverse hardness and higher order contributions, respectively. These heatmaps are plotted against z_h (standard SIDIS z), and q_T (a quantity useful for quantifying transverse momentum hardness relative to Q).

2 Dependencies

This code was written in Python 2.7 using Jupyter Notebook. Before running, be sure to have the following libraries installed

- Bokeh
  - Can install by running (conda install bokeh) in anaconda prompt.
- Holoview
  - Can install by running ("conda install -c pyviz holoviews bokeh")

You must also have the file "utilities.py" in the same directory that you are running the code from. Utilites.py contains all of the function definitions needed to calculate Region ratios. The reason for using bokeh and holoview as our plotting libraries instead of matplotlib, is that matplotlib creates a static png for each region function call. This corresponds to a noticible lag in between any change in the sliders, and updates to the plots. Bokeh and Holoview can create dynamic plots, which allow them to update very quickly while maintaining high resolution in slider variables. They also allow for the use of tools like "hover" which is very useful.

3 How it Works

The first block of code imports the libraries to be used, and uses line magics (%opts) to create a color bar for the created heatmaps, a hover tool which tells you the z_h,q_T and Region value when hovering your cursor over the plot, as well as assigns the colormap style "jet" for the distribution of region values. You can find other colormaps at `http://holoviews.org/user_guide/Colormaps.html`

The code takes the kinematic variables and momentum fractions as inputs in two ways:
1: Explicit inputs

1. k_i (Default = 0.0 GeV)

2. k_f (Default = 0.0 GeV)

3. k_T (Default = 0.2 GeV)

4. ϕ (Default = 0.0 radians)

5. $\text{Mass}_{\text{target}}$ (Default = 0.94 GeV)

6. $\text{Mass}_{\text{produced}}$ (Default = 0.14 GeV)

Example:

```
kinitval = float(raw_input("Enter a value for k_i: ")or 0.0)
```

2: Slider Variables

1. $1 \leq Q \leq 5$

2. $0.1 \leq x_{bj} \leq 1$

3. $0.1 \leq \zeta \leq 1$

4. $0.1 \leq \xi \leq 1$

The only function definition is "r2surf", which is a function of the variables which will become sliders. The explicit inputs are global constants, so there's no need to declare function dependence on them. The two arrays "zhval" and "qtrange" hold N (with N = 75) linearly spaced points ranging from (0.1,1) and (1,5), respectively. We want to evaluate the region functions at all combinations of zhval and qtrange, which was achieved by forming a grid out of them:

```
qtrange, zhval = np.meshgrid(qtrange, zhval, indexing = 'xy')
```

It's easy to see this use when picturing the grid as follows:

$$\begin{bmatrix} qt_1, zh_N & ,qt_2, zh_N & qt_3, zh_N & ... & qt_N, zh_N \\ qt_1, zh_{N-1} & ,qt_2, zh_{N-1} & qt_3, zh_{N-1} & ... & qt_N, zh_{N-1} \\ qt_1, zh_{N-2} & ,qt_2, zh_{N-2} & qt_3, zh_{N-2} & ... & qt_N, zh_{N-2} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ qt_1, zh_1 & ,qt_2, zh_1 & qt_3, zh_1 & ... & qt_N, zh_1 \end{bmatrix}$$

Where each element is then used to calculate Region 2 and 3 for that specific case. These regions depend on values such as x_N and z_N, which needed to be evaluated at each element of the grid defined above. Those values were then stored in the following arrays:

```
zval = sidis.znqt(zhval,xbj,Q,Mpp,Mhh,qtrange) # Nachmann z
xval = sidis.xnac(xbj, Mpp, Q) # Nachmann x
```

where znqt and xnac are defined in the utilities module. Finally, for each point on the grid, and each associated value of x_N and z_N, a corresponding R2 or R3 value could be calculated and stored in the arrays R2list and R3list, respectively. Heatmaps of these regions were created using the Image function from the holoview library (hv.Image()). Holoview pulls data from np.meshgrid() differently than matplotlib, in that the y axis is inverted **without inverting the y tick labels**. This was fixed by using Numpy's "flipud" function with the argument being a region array to invert the y axis independently from the y tick labels. Justification for this can be seen at `https://github.com/ioam/holoviews/issues/155` which also required specifying the bounds of the plot. Bounds on the colorbar, as well as it's label can be set within the Image function as follows:

```
hv.Image(np.flipud(data)), label = "This is your title"\
,bounds = (xi,yi,xf,yf),vdims=hv.Dimension('Color Bar Label', range=(lower
    bound, upper bound)))
```

In the case of returning multiple subplots, you can achieve this by returning the first plot + the second plot. $Plot1 * Plot2$ will overlay two plots instead of creating two separate.

Outside of the function r2surf, kdims defines the label, range, and default values for the slider variables. The function "DynamicMap" is the key to assuring the plots update quickly, and assigns the parameters inside of kdims to be sliders.

4 Before Making Changes

1. If you wish to change things such as the bounds on the independent variables (i.e. zhval), be sure to change the limits on their arrays **AND** the variable "bounds".

2. You can add more sliders simply by incorporating r2surf dependence on them (i.e. r2surf(Q,xbj,zet,xival, **ktval**)). Then, you will need to add information regarding that slider such as its range to kdims. If you are changing an explicit input to a slider variable, be sure to delete the variable being an input.