Transfer Limits Animation

Jovan Bebić

Marija Bebić

June 26, 2017

Abstract

This document is a LaTeX template for creating code documentation. It is typeset *report* document class, which means that it has chapters and sections. The main purpose of the document is to illustrate it's use of figures and listings, but it also showcases cross referencing bibliography entries, etc.

Revision Notes

Authors	Date	Rev	Notes
JZB	Jun 11-14, 2017	Draft	Template for use by others

Contents

List of Figures

List of Listings

Chapter 1

Introduction

This is a template LaTeX file useful when preparing software documentation. It is compiled with pdflatex using included DOS batch file called runAll.bat. In the following report, we are documenting the process of creating a graph needed for a presentation.

1.1 Including and captioning figures

The first step was to create the circle that would be the basis of the graph. This is done in the following lines of code:

Listing 1.1: code/base_frame_v1.py

```
1 th = np.arange(0, 2*np.pi, np.pi/100)
2 x = R*cos(th)
3 y = R*sin(th)
```

Theta(th) is defined by a range from 0 to 2pi, with a step of pi/100. The smaller the step, the more accurate and defined the circle is.

For the purpose of this graph, five areas were created to represent different points throughout the circle. This is done in the following lines of code:

Listing 1.2: code/base_frame_v1.py

```
AreaNames= ['A', 'B', 'C', 'D', 'E']
MaxFlows = np.array([[np.nan, 100. , 100. , np.nan, 100.],
                     [100. , np.nan, 200. , np.nan, 100.],
                     [100. , 200. , np.nan, 100. , np.nan],
                     [np.nan, np.nan, 100. , np.nan, 100.],
                     [100. , 100. , np.nan, 100. , np.nan
                        ]])
```

The reference points, marked in gray, on the circle were created by creating an theta range from 0 to 2pi, with a step of 2pi divided by the area array. This separates the reference points by the area array evenly.

The points on the graph were plotted using the following lines of code:

Listing 1.3: code/base_frame_v1.py

```
nAr = MaxFlows.shape[0] # Number of areas in input data
aStep = np.pi*2/nAr
th1 = np.arange(0, 2*np.pi, aStep)
x1 = R*cos(th1) #x coordinates for ref. angles of area
y1 = R*sin(th1) #y coordinates for ref. angles of area
```

To create the red and blue points on the circle, we needed to first determine if the vector from i to j passes to the right or left relative to center, looking from each gray point individually.

To distribute the red points, we first needed to find the scalar product of the vector from point i to j, and from point i to the origin. This is done in the following lines of code:

Listing 1.4: code/base_frame_v2.py

```
1               Pijx = x1[j] - x1[i] # delta in x
2               Pijy = y1[j] - y1[i] # in y
3               # vector delta from x1[i] to Center, origin at
                    x1[i] tip at Center
4               PiCx = 0 - x1[i] # delta in x
5               PiCy = 0 - y1[i] # in y
```

In the scalar product of these two vectors, the i and j components equal zero; the only component left is k, which is calculated as follows; if the k component was less than zero, it was added to the left tally, and if greater than zero, it was added to the right tally. This is shown in the following lines of code:

Listing 1.5: code/base_frame_v2.py

```
1               # k coordinate of vector product
2               kCom = (Pijx*PiCy) - (Pijy*PiCx)
```

Listing 1.6: code/base_frame_v2.py

```
1               if kCom < 0:
2                   # print('K value points to the left')
3                   left_tally[i] = left_tally[i] + MaxFlows[i,
                        j]
4                   TallyAngles[i,j] = -np.arccos(temp)/np.pi
                        *180.
5               else:
6                   #print('K value points to the right')
7                   right_tally[i] = right_tally[i] + MaxFlows[
                        i,j]
8                   TallyAngles[i,j] = np.arccos(temp)/np.pi
                        *180.
```

To plot the points as (x,y) coordinates, we followed the same formula for the gray points. However, the theta values were slightly different. For

the blue points, the left tally was subtracted, and for the red points, the right tally was added. These values were multipled by our conversion factor flowscale, represented as rad/MW, to keep the angles in radians. This is shown in the following lines of code, for the blue and red points, respectively:

Listing 1.7: code/base_frame_v2.py

```
#left tally
th3 = th1 - left_tally*flowScale
#right tally
th4 = th1 + right_tally*flowScale
```

To plot the arcs between the gray and blue/red points, we used a 4 point Bezier curve with point 1 being the gray point, point 2 and 3 being the 'golden section' of the curve, and point 4 being the red or blue we were trying to connect to. This is shown as follows:

Listing 1.8: code/base_frame_v2.py

```
                mdptx = (x1[i]+x1[j])/2
                mdpty = (y1[i]+y1[j])/2
                print('Midpoint between i and j is %g, %g'
                    %(mdptx, mdpty))

                goldsectx = mdptx/2
                goldsecty = mdpty/2
                print('Golden section point equals: %g, %g'
                     %(goldsectx, goldsecty))

                verts = [
                        (x1[i], y1[i]),
                        (goldsectx, goldsecty),
                        (goldsectx, goldsecty),
                        (x1[j], y1[j])]

                #Bezier Curve beginning
```

```
16
17                  codes = [Path.MOVETO,
18                           Path.CURVE4,
19                           Path.CURVE4,
20                           Path.CURVE4,
21                           ]
```

Because the right and left tallies were organized in arrays and not points,

This plots the gray points shown in the following figure: Fig. 1.1.

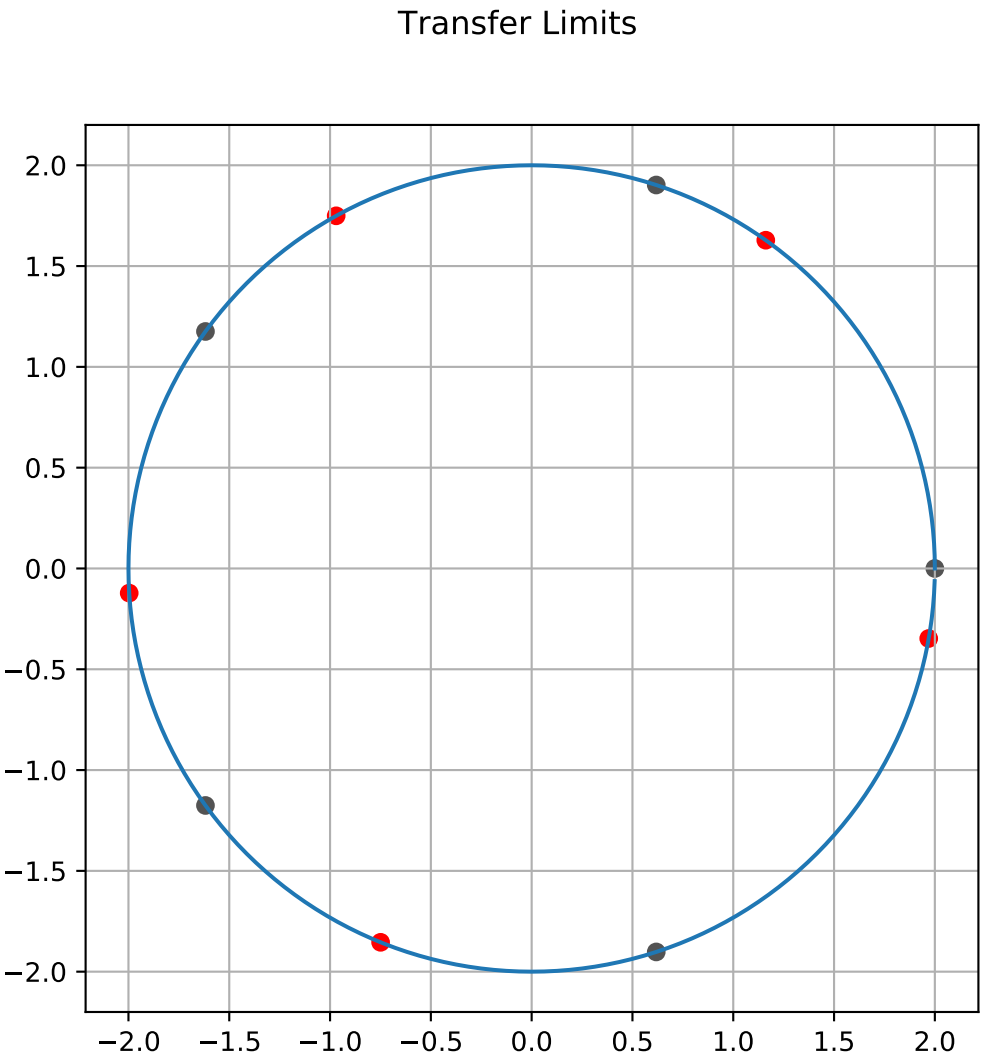

Figure 1.1: Area markers arranged on a circle

1.2 Including and annotating code

Listing 1.9: code/base_frame_v0.py

```
mdptx = (x1[i]+x1[j])/2
mdpty = (y1[i]+y1[j])/2
print('Midpoint between i and j is %g, %g'
    %(mdptx, mdpty))

goldsectx = mdptx/2
goldsecty = mdpty/2
print('Golden section point equals: %g, %g'
     %(goldsectx, goldsecty))

verts = [
        (x1[i], y1[i]),
        (goldsectx, goldsecty),
```

1.3 Other formatting

We saw in 1.1 how to include figures.

1. Preprocess historic AMI data to enable study based on actual measurements.

2. Execute OpenDSS in snapshot mode to review voltage contours at a wide range of operating conditions.

3. Review the resulting patterns and place OpenDSS *monitors* at nodes that experience greatest voltage change.

4. Execute OpenDSS in temporal mode to collect temporal voltage recordings at monitored nodes.

5. Review voltage histograms at monitored buses to quantify the frequency and magnitude of voltage excursions.

In the following sections, we briefly describe the tool-chains that facilitate this process.