```
Script started on Sat 17 Dec 2016 10:19:50 PM CST
bash: wq: command not found
^[]0;mgrobelny@awesomeServer:~/Scripts/github/Data-viz-Circle-plot^G^[]7;file://awesomeServer/home/mgrobelny/Scripts/git
hub/Data-viz-Circle-plot^G^[[?1034h^[[01;31m^B22:19:50 ^[[01;32m^Bmgrobelny ^[[02;36m^BawesomeServer ^[[01;34m^B/home/mg
robelny/Scripts/github/Data-viz-Circle-plot ^[[00;33m^Bmaster^[[00m

$ cat d^Gata_viz.^Gpy ^H^[[K

#import cairocffi as cairo
import cairo
import sys
import getopt
import math
import numpy as np

import matplotlib.pyplot as plt
#from data_viz_functions import *



################################################################################
################################################################################

# create a dictionary for chromosome size
chr_size_dic = {'groupI' : 28185914, 'groupII'   : 23295652, 'groupIII'   : 16798506, 'groupIV'  : 32632948,
'groupIX'   : 20249479, 'groupV'    : 12251397, 'groupVI'    : 17083675, 'groupVII' : 27937443,
'groupVIII' : 19368704, 'groupX'    : 15657440, 'groupXI'    : 16706052, 'groupXII' : 18401067,
'groupXIII' : 20083130, 'groupXIV'  : 15246461, 'groupXIX'   : 20240660, 'groupXV'  : 16198764,
'groupXVI'  : 18115788, 'groupXVII' : 14603141, 'groupXVIII' : 16282716, 'groupXX'  : 19732071,
'groupXXI'  : 11717487}

chrm_name_order_list= ['groupI', 'groupII','groupIII', 'groupIV',
'groupV', 'groupVI', 'groupVII',
'groupVIII', 'groupIX', 'groupX', 'groupXI', 'groupXII',
'groupXIII', 'groupXIV', 'groupXV',
'groupXVI', 'groupXVII', 'groupXVIII', 'groupXIX','groupXX','groupXXI']

stats_dic= {'Fst': (0, 1),'Div' :(0,1)}

stat_list = ['Fst','Div']

color_grad_dic= {'Fst': 'Greens','Div':'seismic'}


#'Rand': ['0,0,0','0.5,0.5,0.1']}


################################################################################
viz_parameters = {'total_genome_size': sum(chr_size_dic.values()),
'number_of_chr': len(chr_size_dic),
'rad_inner' : 250,
'ring_gap': 10,
'arc_padding_in_degrees': 2,
'last_degree_end': 0,
'ring_width': 35,
'total_degrees': 0,
'10mb_step_off_set':32,
'font_size': 5, # need to test sizes
'width': 2,   # must be float ? need to test
'dash_pattern': [3,1], # a sequence specifying alternate lengths of on and off stroke portions.
'label_units': "Mb",
'key_loc_offset' : 90,
'key_width': 30,
'key_height' : 120,
'key_sep_distance':14,
'key_degree_off_set': -10,
"key_label_font_x": 0.7

#'fill_color' :'0.4,0.4,0.4' ,
#'trim_color' : '0,0,0'
}

# calculate number of degrees per nucleotide
viz_parameters['degree_per_nuc'] = float(360 - (viz_parameters['number_of_chr'] * viz_parameters['arc_padding_in_degrees
'])) / float(viz_parameters['total_genome_size'])

img = {}
img['height']     = 800
img['width']      = 800
img['center_x']   = img['width'] / 2.0
img['center_y']   = img['height'] / 2.0
img['font_size']  = 16

################################################################################
# available colors specturms names
cmaps = [('Perceptually Uniform Sequential',
                        ['viridis', 'inferno', 'plasma', 'magma']),
     ('Sequential',     ['Blues', 'BuGn', 'BuPu',
                         'GnBu', 'Greens', 'Greys', 'Oranges', 'OrRd',
                         'PuBu', 'PuBuGn', 'PuRd', 'Purples', 'RdPu',
                         'Reds', 'YlGn', 'YlGnBu', 'YlOrBr', 'YlOrRd']),
     ('Sequential (2)', ['afmhot', 'autumn', 'bone', 'cool',
                         'copper', 'gist_heat', 'gray', 'hot',
                         'pink', 'spring', 'summer', 'winter']),
     ('Diverging',      ['BrBG', 'bwr', 'coolwarm', 'PiYG', 'PRGn', 'PuOr',
                         'RdBu', 'RdGy', 'RdYlBu', 'RdYlGn', 'Spectral',
                         'seismic']),
     ('Qualitative',    ['Accent', 'Dark2', 'Paired', 'Pastel1',
                         'Pastel2', 'Set1', 'Set2', 'Set3']),
     ('Miscellaneous',  ['gist_earth', 'terrain', 'ocean', 'gist_stern',
                         'brg', 'CMRmap', 'cubehelix',
                         'gnuplot', 'gnuplot2', 'gist_ncar',
                         'nipy_spectral', 'jet', 'rainbow',
                         'gist_rainbow', 'hsv', 'flag', 'prism'])]
# http://matplotlib.org/examples/color/colormaps_reference.html

color_stat_mapper_dic ={}

for stat in stat_list:
    color_stat_mapper_dic[stat] = {}
```

```
################################################################################
#
# Define the path our file
#
#outpath = '/home/a-m/ib501_stud12/shell/data_viz/data_viz.pdf'
outpath='/home/mgrobelny/Scripts/github/Data-viz-Circle-plot/data_viz.pdf'
ps = cairo.PDFSurface(str(outpath), float(img['height']), float(img['width']))
cr = cairo.Context(ps)

# import drawing functions

################################################################################
# # default parameters

# argv = sys.argv[1:]
# try:
#     opts, args = getopt.getopt(argv, "hs:")
# except getopt.GetoptError:
#     print 'kspec.py -k <kmer_size> -x <x_axis_max> -t <type>[fasta|fastq] -f <inputfile>'
#     sys.exit(2)
# for opt, arg in opts:
#     if opt == '-h':
#         print "#--- K-mer frequency graphing script ---#\n"
#         print "Usage:"
#         print 'kspec.py -k <kmer_size> -x <x_axis_max> -t <type>[fasta|fastq] -f <inputfile> \n'
#         print "Goals:"
#         print "1) Take in fastq file and kmerize it and output kmer occurence frequnecy"
#         print "2) Output graph of kmer occurence frequnecy"
#         print "3) Output kmer occurence frequnecy to .tsv file"
#         print "\n"
#
#         sys.exit()
#     elif opt in ("-k"):
#         kmer = arg
#     elif opt in ("-x"):
#         xmax = arg
#     elif opt in ("-f"):
#         file_name = arg
#     elif opt in ("-t"):
#         file_type = arg
# print "Input file:", file_name
# print "Input file type:", file_type
# print "Kmer size:", kmer
# print "X-axis max kmer count:", xmax
# print " "
#
#
# ################################################################################
# # Progress bar is not my own work from:
# # https://gist.github.com/vladignatyev/06860ec2040cb497f0f3
# #
# def progress(count, total, suffix=''):
#     bar_len = 60
#     filled_len = int(round(bar_len * count / float(total)))
#
#     percents = round(100.0 * count / float(total), 1)
#     bar = '=' * filled_len + '-' * (bar_len - filled_len)
#
#     sys.stdout.write('[%s] %s%s ...%s\r' % (bar, percents, '%', suffix))
#     sys.stdout.flush()
#

################################################################################
#
# Convert a radius and a span of degrees into X, Y coordinates #
def get_x_y_coordinates(center_x, center_y, degree, radius):
    if degree <= 90:
        theta = float(degree)
        opp_side = radius * math.sin(math.radians(theta))
        adj_side = radius * math.cos(math.radians(theta))
        x = center_x + adj_side
        y = center_y + opp_side
    elif degree <= 180:
        theta = float(degree - 90.0)
        opp_side = radius * math.sin(math.radians(theta))
        adj_side = radius * math.cos(math.radians(theta))
        x = center_x - opp_side
        y = center_y + adj_side
    elif degree <= 270:
        theta = float(degree - 180.0)
        opp_side = radius * math.sin(math.radians(theta))
        adj_side = radius * math.cos(math.radians(theta))
        x = center_x - adj_side
        y = center_y - opp_side

    else:
        theta = float(degree - 270.0)
        opp_side = radius * math.sin(math.radians(theta))
        adj_side = radius * math.cos(math.radians(theta))
        x = center_x + opp_side
        y = center_y - adj_side
    return (x, y)

# int to roman not my code from:
# https://www.safaribooksonline.com/library/view/python-cookbook/0596001673/ch03s24.html
def int_to_roman(input):
    """ Convert an integer to a Roman numeral. """
    if not isinstance(input, type(1)):
        raise TypeError, "expected integer, got %s" % type(input)
    if not 0 < input < 4000:
        raise ValueError, "Argument must be between 1 and 3999"
    ints = (1000, 900,  500, 400, 100,  90, 50,  40, 10,  9,   5,  4,   1)
    nums = ('M',  'CM', 'D', 'CD','C', 'XC','L','XL','X','IX','V','IV','I')
    result = []
    for i in range(len(ints)):
        count = int(input / ints[i])
        result.append(nums[i] * count)
        input -= ints[i] * count
    return ''.join(result)

################################################################################
```

```
# Data norm fucntion
# Normalizes a stat list value from 0 to 1
def data_norm(chrm_name, chrm_bp_st_dic, list_of_bp_n_stats, type_of_norm):
    stat_val_list =[]
    length_of_list = int(len(list_of_bp_n_stats))
    min_stat_val = 0
    max_stat_val = 0
    for i in range(length_of_list):
        stat_val_list.append(float(list_of_bp_n_stats[i][1]))

    min_stat_val = min(stat_val_list)
    max_stat_val = max(stat_val_list)

    # normalize from 0 to 1
    for i in range(length_of_list):
        if type_of_norm == "log":
            log_min_stat_val= math.log10(min_stat_val)
            log_max_stat_val= math.log10(max_stat_val)
            stat_val= float((math.log10(stat_val_list[i]) − log_min_stat_val)/(log_max_stat_val − log_min_stat_val))

        elif type_of_norm == "norm":
            stat_val= (stat_val_list[i]−min_stat_val)/(max_stat_val−min_stat_val)

        chrm_bp_st_dic[chrm_name][i].append(stat_val)

def stat_to_color(stat, type_of_norm,reverse):

    # create color list based on color group
    color = color_grad_dic[stat]
    number_of_color_breaks = viz_parameters[’key_height’]
    color_list = 0
    color_code = "color_list = plt.cm.%s(np.linspace(0, 1, number_of_color_breaks))" % (color)
    exec(color_code)

    min_stat_val = 0
    max_stat_val = viz_parameters[’key_height’]
    log_min_stat_val= 0
    log_max_stat_val= math.log10(viz_parameters[’key_height’])

    stat_numer_count = 1

    # save color to normalized pixel space values between 1 and 0
    if reverse == ’True’:
        for color in reversed(color_list):
            if type_of_norm == "log":
                # normalize from 0 to 1
                normalized_val= float((math.log10(stat_numer_count) − log_min_stat_val)/float(log_max_stat_val − log_min
_stat_val))

            elif type_of_norm == "norm":
                normalized_val= float(stat_numer_count−min_stat_val)/float(max_stat_val−min_stat_val)
            color_stat_mapper_dic[stat][normalized_val] =  color

            stat_numer_count = stat_numer_count + 1
    elif reverse == ’False’:
        for color in color_list:
            if type_of_norm == "log":
                # normalize from 0 to 1
                normalized_val= float((math.log10(stat_numer_count) − log_min_stat_val)/float(log_max_stat_val − log_min
_stat_val))

            elif type_of_norm == "norm":
                normalized_val= float(stat_numer_count−min_stat_val)/float(max_stat_val−min_stat_val)
            color_stat_mapper_dic[stat][normalized_val] =  color

            stat_numer_count = stat_numer_count + 1


# --------  Drawing functions -------- #

# Draw a circle of arcs based on a list of chr which corresponed to the chrm size dic
def draw_label(text, x, y, font_size,working_degree):

    # Font
    cr.select_font_face("Sans", cairo.FONT_SLANT_NORMAL, cairo.FONT_WEIGHT_NORMAL)
    # Set the font size
    cr.set_font_size(font_size)
    # font color
    cr.set_source_rgb(0, 0, 0)

    # Get the size of the text we want to write, returns a tuple:
    #   (x, y, width, height, dx, dy)
    #
    textents = cr.text_extents(text)
    text_width = textents[2]
    text_height = textents[3]
    #
    # Where you want to draw text may need to be adjusted,
    # depending on the size of the text.
    if working_degree >=100 and working_degree <= 280:
        centered_x= x
        centered_y = y + text_height/2
    else:
        centered_x= x − text_width
        centered_y =y + 1

    cr.move_to(centered_x, centered_y)
    # cr.move_to(x,y)
    cr.show_text(text)

# Draw 10mb label markers
def draw_10mb_labels(chrm_list, level):

    for chrm_name_it in chrm_list:

        break_size = 5000000 #bases
        # find how many 10mb breaks there are for chrm_name
        five_mb_break = int(chr_size_dic[chrm_name_it] / break_size)

        # determine degree of 10mb step line
        five_mb_step_degree = float(break_size * viz_parameters[’degree_per_nuc’])
```

```
        # for i number of breaks draw a line every 5mb and a
        for i in range(1,five_mb_break+1):

            # calculate 5mb step
            working_degree = five_mb_step_degree * i + viz_parameters[’last_degree_end’]

            # find the x and y pos of the location of the 10mb step
            sx, sy = get_x_y_coordinates(img[’center_x’], img[’center_y’], working_degree, viz_parameters[’rad_inner’]
− viz_parameters[’10mb_step_off_set’]* 1.4)

            # move to that location
            cr.move_to(sx, sy)

            # find the end of the line
            end_of_line = viz_parameters[’rad_inner’] + viz_parameters[’ring_gap’] * level + viz_parameters[’ring_width’
] * (level −1)
            sx, sy = get_x_y_coordinates(img[’center_x’], img[’center_y’], working_degree, end_of_line)
            # write line
            cr.line_to(sx, sy)

            # made dashed line
            cr.set_dash(viz_parameters[’dash_pattern’])

            if i % 2 == 0:
                # darker grey line color
                cr.set_source_rgb(0.2, 0.2, 0.2)

                # stroke a thicker line
                cr.set_line_width(viz_parameters[’width’] + 0.1)

                cr.stroke()

                # find the x and y location for the 10m label
                label_x, label_y = get_x_y_coordinates(img[’center_x’], img[’center_y’], working_degree, viz_parameters[
’rad_inner’]  − viz_parameters[’10mb_step_off_set’]* 1.49)
                #working_degree, viz_parameters[’rad_inner’] − viz_parameters[’10mb_step_off_set’]*1.75)

                # write label name w/ units
                label = str(int(i * 5)) + viz_parameters[’label_units’]

                # pass to draw label function
                draw_label(label, label_x, label_y, 7, working_degree)

            else:
                # ligher grey line color
                cr.set_source_rgb(0.4, 0.4, 0.4)

                # stroke a thinner line
                cr.set_line_width(viz_parameters[’width’]−1)
                cr.stroke()

        # Update where the start of next chrm is os labeling can be indexed corretly
        viz_parameters[’total_degrees’] = float(viz_parameters[’degree_per_nuc’]) * float(chr_size_dic[chrm_name_it])
        viz_parameters[’last_degree_end’] = float(viz_parameters[’last_degree_end’]) + float(viz_parameters[’total_degre
es’]) + float(viz_parameters[’arc_padding_in_degrees’])

# Draw chrm name labels use ither provided labels from list or generate new names with roman number (roman= 1)
def chrm_label(chrm_list, total_levels, roman):
    count = 0
    for chrm_name in chrm_list:
        degree_for_label = float(viz_parameters[’degree_per_nuc’]) * float(chr_size_dic[chrm_name])/2
        working_degree = viz_parameters[’last_degree_end’] + degree_for_label
        radian_for_label = viz_parameters[’rad_inner’] + viz_parameters[’ring_gap’] * total_levels + viz_parameters[’rin
g_width’] * total_levels

        label_x, label_y = get_x_y_coordinates(img[’center_x’], img[’center_y’], working_degree, radian_for_label)
        #working_degree, viz_parameters[’rad_inner’] − viz_parameters[’10mb_step_off_set’]*1.75)
        label = ""

        if roman == 1:
            label = str(int_to_roman(count + 1))
            count = count +1
        else:
            label = str(chrm_name)

        # pass to draw label function
        draw_label(label, label_x, label_y, 12, working_degree)


        # Update where the start of next chrm is os labeling can be indexed corretly
        viz_parameters[’total_degrees’] = float(viz_parameters[’degree_per_nuc’]) * float(chr_size_dic[chrm_name])
        viz_parameters[’last_degree_end’] = float(viz_parameters[’last_degree_end’]) + float(viz_parameters[’total_degre
es’]) + float(viz_parameters[’arc_padding_in_degrees’])

def color_key(total_levels,location,trim): #min, max,color_start, color_end,
    if location == "top_left":
        working_degree_key = 225
    elif location == "bottom_right":
        working_degree_key  = 45
    elif location == "bottom_left":
        working_degree_key  = 135
    else:
        # default to top_right location for key
        working_degree_key  = 315

    radius_key = viz_parameters[’rad_inner’] + viz_parameters[’ring_gap’] * total_levels + viz_parameters[’ring_width’]
* total_levels + viz_parameters[’key_loc_offset’]
    sx, sy = get_x_y_coordinates(img[’center_x’], img[’center_y’], working_degree_key + viz_parameters[’key_degree_off_s
et’], radius_key)
    sx_key = 0
    for i in range(total_levels):

        sx_key = sx + (viz_parameters[’key_width’])* i +viz_parameters[’key_sep_distance’] * i

        ############################
        # Add black trim to key 0 no , 1 yes

        if trim == 0:
            # fill with color gradient
            # Pick stat based on level
            key = stat_list[i]
            it_y = sy
            # sort all norm values and color each line basaed on the dictionary of color

            for norm_val in reversed(sorted(color_stat_mapper_dic[key].keys())):
                color = color_stat_mapper_dic[key][norm_val]

                cr.set_source_rgba(color[0], color[1], color[2], color[3])
                cr.move_to(sx_key, it_y)
                cr.line_to(sx_key + viz_parameters[’key_width’],it_y)
                cr.set_dash([])
                cr.stroke()

                # update the y pos of next line
                it_y = it_y + +1

            # Draw Key labels

            # Draw stat name above key
            draw_label(stat_list[i], sx_key + viz_parameters[’key_width’]/2 +4, sy-3, 12* viz_parameters[’key_label_font_x’],working_degree_key)

            # Draw min labels
            draw_label(str(stats_dic[stat_list[i]][0]), sx_key-2, sy + viz_parameters[’key_height’], 9* viz_parameters[’key_label_font_x’],working_degree_key)

            # Draw max label
            draw_label(str(stats_dic[stat_list[i]][1]), sx_key- 2, sy, 9* viz_parameters[’key_label_font_x’],working_degree_key)


        else:
            #cr.set_source_rgb(viz_parameters[’trim_color’])

            #  trim in black
            cr.move_to(sx_key, sy)
            cr.rectangle(sx_key, sy, viz_parameters[’key_width’], viz_parameters[’key_height’])
            cr.close_path()
            cr.set_line_width(viz_parameters[’width’] - 1)
            cr.set_dash([])
            cr.set_source_rgb(0, 0, 0)
            cr.stroke()

# Draw chrm arc for a given level w (1) or wo (0) balck trim
def chrm_arc(chrm_name, level, trim):
    # Create intial arc
    radius = viz_parameters[’rad_inner’] + viz_parameters[’ring_gap’] * level + viz_parameters[’ring_width’] * level

    # calculate the number of degrees the are will span based on chrm size
    viz_parameters[’total_degrees’] = float(viz_parameters[’degree_per_nuc’]) * float(chr_size_dic[chrm_name])

    # draw first arc based on the ending of the pervious arc
    sx, sy = get_x_y_coordinates(img[’center_x’], img[’center_y’], viz_parameters[’last_degree_end’], radius)

    cr.move_to(sx, sy)
    start_deg = viz_parameters[’last_degree_end’]
    end_deg = viz_parameters[’last_degree_end’] + viz_parameters[’total_degrees’]

    # draw outer arc
    cr.new_sub_path()
    cr.arc_negative(img[’center_x’], img[’center_y’], radius, math.radians(end_deg),math.radians(start_deg))

    # draw line to inner arc
    sx, sy = get_x_y_coordinates(img[’center_x’], img[’center_y’], viz_parameters[’last_degree_end’], radius - viz_parameters[’ring_width’])
    cr.line_to(sx, sy)

    # draw reverse arc
    cr.arc(img[’center_x’], img[’center_y’], radius - viz_parameters[’ring_width’], math.radians(start_deg), math.radians(end_deg))

    cr.close_path()
    cr.set_line_width(viz_parameters[’width’] - 1)
    cr.set_dash([])
    ############################
    # Add black trim to chrm arcs 0 no , 1 yes
    if trim == 0:
        pass
        # #fill with grey color
        # if level ==1:
        #     cr.set_source_rgb(0.4, 0.4, 0.4)
        #     cr.fill()
        # if level ==0 :
        #     closest_val = min(sorted(color_stat_mapper_dic[stat_list[0]].keys()), key=lambda x:abs(x-float(0)))
        #     color = color_stat_mapper_dic[stat_list[level]][closest_val]
        #     cr.set_source_rgba(color[0], color[1], color[2], color[3])
        #     cr.fill()
    else:
        #cr.set_source_rgb(viz_parameters[’trim_color’])

        #  trim in black
        cr.set_source_rgba(0, 0, 0)
        cr.stroke()

    ############################

    # Update the end of viz parameter[last_degree_end] + padding --> for next arc start degree
    viz_parameters[’last_degree_end’] = float(viz_parameters[’last_degree_end’]) + float(viz_parameters[’total_degrees’]) + float(viz_parameters[’arc_padding_in_degrees’])

# Draw all chrm arc for a given level w (1) or wo (0) balck trim
def draw_chrom_arc(chrm_list, level, trim):
    for chrm_name in chrm_list:
        chrm_arc(chrm_name, level, trim)
    viz_parameters[’last_degree_end’] = 0


# Data drawing fucntion


def draw_stats(chrm_list, level):
```

```
    for chrm_name in chrm_list:
        work_dic = level_to_dic[level]
        radius_stat = viz_parameters[’rad_inner’] + viz_parameters[’ring_gap’] * level + viz_parameters[’ring_width’] * level

        # Create intial arc
        # calculate the number of degrees the are will span based on chrm size
        viz_parameters[’total_degrees’] = float(viz_parameters[’degree_per_nuc’]) * float(chr_size_dic[chrm_name])


        arc_length= float(2 * math.pi * radius_stat*( viz_parameters[’total_degrees’] / 360))

        degrees_per_pixel = float(viz_parameters[’total_degrees’]/ arc_length)
        window_size_for_smoothing = float(chr_size_dic[chrm_name] / arc_length)

        # Data smoothing
        window_bp_counter = window_size_for_smoothing
        windowed_stats = []
        smoothed_stats = [] # per pixel
        for list_it in range(len(work_dic[chrm_name])):
            if level ==1:
                if  int(window_bp_counter) >= int(work_dic[chrm_name][list_it][0]):
                    windowed_stats.append(work_dic[chrm_name][list_it][-1])

                elif int(work_dic[chrm_name][list_it][0]) >= int(window_bp_counter) and len(windowed_stats) == 0:
                    window_bp_counter = window_bp_counter + window_size_for_smoothing
                    smoothed_stats.append(0)
                    windowed_stats = []
                else:
                    smoothed_stats.append(np.mean(windowed_stats))
                    window_bp_counter = window_bp_counter + window_size_for_smoothing
                    windowed_stats = []
                working_degree = viz_parameters[’last_degree_end’]
                for smoothed_stat in smoothed_stats:

                    sx, sy = get_x_y_coordinates(img[’center_x’], img[’center_y’], working_degree, radius_stat)

                    cr.move_to(sx, sy)

                    line_x, line_y = get_x_y_coordinates(img[’center_x’], img[’center_y’], working_degree, radius_stat - viz_parameters[’ring_width’])
                    cr.line_to(line_x, line_y)

                    cr.set_dash([])
                    cr.set_line_width(1.25)


                    # Find color value closest to the input stat value
                    # following line is not my code from : http://stackoverflow.com/questions/12141150/from-list-of-integers-get-number-closest-to-a-given-value
                    closest_val = min(sorted(color_stat_mapper_dic[stat_list[level]].keys()), key=lambda x:abs(x-float(smoothed_stat)))

                    color = color_stat_mapper_dic[stat_list[level]][closest_val]
                    cr.set_source_rgba(color[0], color[1], color[2], color[3])
                    cr.stroke()

                    working_degree = float(working_degree + degrees_per_pixel)

            if level ==0:

                degree_for_data_pt = float(work_dic[chrm_name][list_it][0]) *viz_parameters[’degree_per_nuc’]

                stat_for_drawing = work_dic[chrm_name][list_it][-1]
                start_deg = viz_parameters[’last_degree_end’]
                working_degree = start_deg + degree_for_data_pt

                sx, sy = get_x_y_coordinates(img[’center_x’], img[’center_y’], working_degree, radius_stat)

                cr.move_to(sx, sy)

                line_x, line_y = get_x_y_coordinates(img[’center_x’], img[’center_y’], working_degree, radius_stat - viz_parameters[’ring_width’])
                cr.line_to(line_x, line_y)

                cr.set_dash([])
                cr.set_line_width(2)


                # Find color value closest to the input stat value
                # following line is not my code from : http://stackoverflow.com/questions/12141150/from-list-of-integers-get-number-closest-to-a-given-value
                closest_val = min(sorted(color_stat_mapper_dic[stat_list[level]].keys()), key=lambda x:abs(x-float(stat_for_drawing)))

                color = color_stat_mapper_dic[stat_list[level]][closest_val]

                cr.set_source_rgba(color[0], color[1], color[2], color[3])
                cr.stroke()

        # Update the end of viz parameter[last_degree_end] + padding --> for next arc start degree

        viz_parameters[’last_degree_end’] = float(viz_parameters[’last_degree_end’]) + float(viz_parameters[’total_degrees’]) + float(viz_parameters[’arc_padding_in_degrees’])
    print "Done with level:", level
# -------- Main Drawing function --------- #
# Draw all chrm arc for a given level w (1) or wo (0) balck trim w 10mb labels, color
def draw_chrom_arc_w_label(chrm_list, total_levels, trim, roman, location):
    viz_parameters[’last_degree_end’] = 0

    # draw all breaks and labels
    draw_10mb_labels(chrm_list, total_levels)
    viz_parameters[’last_degree_end’] = 0
    # draw chm labels
    chrm_label(chrm_list, total_levels, roman)
    viz_parameters[’last_degree_end’] = 0
    #Draw all chrm arcs

    if trim == 0:
        for i in range(total_levels):
            viz_parameters[’last_degree_end’] = 0
            draw_stats(chrm_list, i)

            viz_parameters[’last_degree_end’] = 0
            draw_chrom_arc(chrm_list, i, 0)
            color_key(total_levels, location, 0)
    else:
        for i in range(total_levels):

            viz_parameters[’last_degree_end’] = 0
            draw_chrom_arc(chrm_list, i, 0)
            viz_parameters[’last_degree_end’] = 0
            draw_stats(chrm_list, i)
            viz_parameters[’last_degree_end’] = 0
            draw_chrom_arc(chrm_list, i, 1)

            color_key(total_levels, location, 0)
            color_key(total_levels, location, 1)


###############################################################################
# # Data import

# Create fst data dictionary
fst_stats = {}
rna_stats = {}
for chrm_name in chrm_name_order_list:
    fst_stats[chrm_name] = []
    rna_stats[chrm_name] = []

level_to_dic ={0: fst_stats,
1 :rna_stats }

# Add each chromosome to the dictionary and store the
# basepair and statistical value
# add each chrm, bp and stat pt to dictionary

# Open fst Data file
in_file = ’./Pop_fst_out.tsv’
fh1_fst_file = open(in_file, ’r’)

# skip header
next(fh1_fst_file)
for line in fh1_fst_file:
    #strip new line char
    line = line.strip(’\n’)
    # remove spaces
    line = line.replace(" ", "")
    # split tabs
    line = line.split(’\t’)
    # append data to each dictionary of  list
    fst_stats[line[0]].append([line[1],line[2]])
fh1_fst_file.close

#repeate for Div data_viz
in_file2 = ’./Pop_div_data.tsv’
fh2_Div_file = open(in_file2, ’r’)

# skip header
next(fh2_Div_file)
for line in fh2_Div_file:
    #strip new line char
    line = line.strip(’\n’)
    # remove spaces
    line = line.replace(" ", "")
    # split tabs
    line = line.split(’\t’)
    # append data to each dictionary of list
    rna_stats[line[0]].append([line[1],line[2]])
fh2_Div_file.close

#  Thus:
#  For rna_stats[’chrmII’] outputs ["bp","stat"]
#  rna_stats[’chrmII’][0] = ["bp","stat"]
#  rna_stats[’chrmII’][0][0] = Base pair
#  rna_stats[’chrmII’][0][1] = stats

###############################################################################
# Data normalization
for chrm_name in chrm_name_order_list:
    #data_norm(chrm_name,fst_stats,fst_stats[chrm_name], "norm")
    data_norm(chrm_name,rna_stats,rna_stats[chrm_name], "log")


###############################################################################
# Map colors to stat

stat_to_color(’Fst’, "norm",’False’)
stat_to_color(’Div’,"norm","False")


###############################################################################
# Draw final image
draw_chrom_arc_w_label(chrm_name_order_list, 2, 1, 1,"def")
###############################################################################

#
# End of file
#
# Close the file
#
cr.show_page()
^[]0;mgrobelny@awesomeServer:~/Scripts/github/Data−viz−Circle−plot^G^[]7;file://awesomeServer/home/mgrobelny/Scripts/git
hub/Data−viz−Circle−plot^G^[[01;31m^B22:20:07 ^[[01;32m^Bmgrobelny ^[[02;36m^BawesomeServer ^[[01;34m^B/home/mgrobelny/S
cripts/github/Data−viz−Circle−plot ^[[00;33m^Bmaster^[[00m

$ python data^G_viz.p^Gy
Done with level: 0
Done with level: 1
^[]0;mgrobelny@awesomeServer:~/Scripts/github/Data−viz−Circle−plot^G^[]7;file://awesomeServer/home/mgrobelny/Scripts/git
hub/Data−viz−Circle−plot^G^[[01;31m^B22:21:20 ^[[01;32m^Bmgrobelny ^[[02;36m^BawesomeServer ^[[01;34m^B/home/mgrobelny/S
```

```
cripts/github/Data−viz−Circle−plot ^[[00;33m^Bmaster^[[00m

$ exit
exit

Script done on Sat 17 Dec 2016 10:21:30 PM CST
```