APPENDIX 1 For

How Good is ‘Good Enough?’ Major Element Chemical Analyses of Planetary Basalts by Spacecraft Instruments.

Allan H. Treiman\*1, Justin Filiberto1, Edgard Rivera-Valentín1

#  Planetary Science Journal, 2020.

# Appendix 1. Graphing and Contouring Routine

# Appendix 1 for this article is the Python code used to generate the contour plots in the manuscript. The code, given below, is built on public-domain functions and routines. The only novel code is the block where the values for the 67% and 95% contours (1- and 2- of the distributions) are derived. The code was run in a Jupyter notebook; graphs produced there were exported, and then cleaned up and annotated in either Photoshop© or Deltagraph©.

*# Libraries*

import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import kde

*# Import data from .csv file. Here, thousands of points from XXX.csv file;*

*# change file name to fit your data.*

data = np.loadtxt(XXX.csv',delimiter=',',skiprows=0)

*# for the kde function, the data may need to be transposed*

datax, datay = data.T

*# Create a figure with 1 plot area*

fig, axes = plt.subplots(ncols=1, nrows=1, figsize=(18, 18))

*# Figsize is size of full figure in inches x inches, adjust at will*

#

*# Evaluate a gaussian kde on a regular grid of nbins x nbins over plot extents;*

*# Set nbins at will*

nbins = 250

k = kde.gaussian\_kde(data.T)

datax\_i, datay\_i = np.mgrid[35:75:nbins\*1j, 0:9:nbins\*1j]

*# to plot over data extents, rather than prescribed limits,*

*# invoking = np.mgrid[x.min();x.max();nbins\*1j, y.min():y.max():nbins\*1j]*

*# where x.min() is the minimum values on the X axis, x.max() is th maximum value*

*# on the X axis, etc.*

*# Set the axis values at will*

*#*

dataz\_i = k(np.vstack([datax\_i.flatten(), datay\_i.flatten()]))

*# dataz\_i is an array of [nbins \* nbins] of the kde-computed point densities;*

*# it is what actually gets plotted*

#

*# this Block calculates 1-sigma and 2-sigma values for the distribution*

*# First, calculate parameters of the distribution, length, max, and sum.*

dataz\_length = len(dataz\_i) *#in function form*

dataz\_max = max(dataz\_i) *#in function form*

dataz\_sum = sum(dataz\_i) *#in function form*

*#*

*# Then sort the list of all values in numerical order, 0 to max(zi), as zj!*

dataz\_sort=sorted(dataz\_i)

*#*

*# Then, starting from high end of ordering, start subtracting values one by one,*

*# until the value goes below 1-sigma, i.e. so that 68% of the values are above the cutoff.*

*# And quit by break.*

dzisum=dataz\_sum

for n in range(62499,0,-1):

 dzisum=dzisum-dataz\_sort[n]

 if dzisum<(dataz\_sum\*0.32):

 dataz\_onesigma=dataz\_sort[n]

 break

#

*# Then starting from high end of ordering, start subtracting values one by one,*

*# until the value goes below 2-sigma, i.e. so that 95% of the values are above the cutoff.*

*# And quit by break.*

dzisum=dataz\_sum

for n in range(62499,0,-1):

 dzisum=dzisum-dataz\_sort[n]

 if dzisum<(dataz\_sum\*0.05):

 dataz\_twosigma=dataz\_sort[n]

 break

*## end of standard deviations block*

#

*# contour plot, with graded colours*

axes.set\_title('Contour')

axes.set\_xlim(35, 75)

axes.set\_ylim(0, 9)

# set the axis limits at will

#

*# set contour levels to: two-sigma level, one-sigma level, and*

*# just below the maximum value, which makes a dot at the maximum value*

levels = [dataz\_twosigma, dataz\_onesigma, (0.999\*dataz\_max)]

axes.pcolormesh(datax\_i, datay\_i, dataz\_i.reshape(datax\_i.shape), shading='gouraud', cmap=plt.cm.BuGn)

axes.contour(datax\_i, datay\_i, dataz\_i.reshape(datax\_i.shape), levels )

*# the command "levels" indicates the values for the contours, over-riding the automatic*

*# contouring option.*