Testing a Lunar Crater Field for Spatial Randomness Using Kolmogorov-Smirnov.

Kurt A. Fisher and Russell Fuller, Undergraduates
University of Utah

April 19, 2013

Abstract

Two procedures were executed in support of a research project concerning the absolute modeling age dating of lunar crater Hell Q. First, the spatial distribution of 890 small lunar craters binned into six 2^n diameter classes were each tested for position randomness by applying the Kolmogorov-Smirnov (KS) test. Results indicate that three of five crater bins counted in the observation field are not contaminated with secondary impact craters to a statistically significant $\alpha = 0.10$ level. Computer aided KS test computations were inconclusive as to two crater bin classes, but future work will resolve those results by manual recomputation. Second, in aid of further testing of the field's spatial randomness using specialty astrogeologic software, multivariable linear regression was used to determine translations equations for the orthogonal coordinates of the craters in the observing field and to convert those coordinates into the selenographic coordinate system. This applied statistics problem illustrates the difficulty of determining for any abstract field of seemingly random points, if the points are in fact spatially distributed randomly.

Introduction

This matter involves executing two procedures in support of an ongoing astrogeology research project, and in the interests of brevity, a detailed description of that project is omitted here.

1 Task 1: Test the spatial randomness of a set of points in a finite R^2 field.

1.1 Summary and Conclusion

The first supporting procedure involved testing the spatial randomness of 890 small lunar craters binned into six 2^n diameter classes by applying the

Kolmogorov-Smirnov (KS) test, and in the KS test, the distribution of k 2nd nearest neighbor distances in an observation field is compared to the k2nn distances in a same dimensioned reference distiibution computed from a simulated random field. The null hypothesis was that the craters binned into six 2^n diameter classes were spatially distributed randomnly, and our alternative hypothesis was that the craters in each class were non-randomly distributed to a level of significance of $\alpha = 0.10$. If the KS test p value was less than the critical value of $\alpha = 0.10$, the field is non-random. Table 1, p. 14, lists the results of KS test for five of the binned fields, and Figure 1, p. 17, shows a plot of the distribution of observation field of k2nn distances, standardized as empircal continuous distributions, compared to a standardized distribution of k2nn distances in a same dimensioned field created by random number simulation.

For three of five diameter bins, the alternative hypothesis was rejected, and the null hypothesis of spatial randomness was accepted, but the KS test result, computation of which was assisted by computer software R, was inconsistent, and future work will resolve the ambiguity of those results by manual computation of those bins' KS p-value. Additionally, the rejection of the two crater diameter bins for randomness may indicate that those crater bins are contaminated by secondary impacts. Future work also will evaluate the geologic setting and features of those craters for secondary impact characteristics. Geologic assessment may result in the exclusion of some craters from the two bins initially found to be non-random.

1.2 Methods

The Kolmogorov-Smirnov test is one of several tests for spatial randomness of points in a field based on the kth nearest neighbor distances measure[3-4]. *See* [5] for a current overview of spatial randomness practice. In the k2nn distance process, the cartesian pairs of points in an observing field are measured, and

then the distance between each point and its respective second nearest neighbor are found. A collection the the k2nn distances for each point in a field has a histogram distribution that if right-skewed represents a field that is more ordered than a random field, and if the distribution is left-skewed then the field is less ordered than a random field. A right-skewed k2nn distance distribution implies that points are closer together than a normally distributed field, and a left-skewed k2nn distance distribution implies that the points are farther apart relative to a normally distributed field.

However, the spatial randomness of an observing frame is not determined with respect to a normal random distribution because a set of points that are spatially distributed randomly inside a *finite* field will not have normally distributed k2nn distances. The k2nn distances in a finite simulated spatially random field random field will have a skewed, non-normal distribution. Conversely, if an infinite sized field, a histogram of k2nn distances are spatially *distributed normally*. This counter-intuitive property of finite counting fields comes from the loss of information regarding the true k second nearest neighbor that is outside the observing frame, and the true k2nn is replaced with a point inside the observing frame, Figure 2, p. 18. The non-randomness of random finite spatial fields prevents the use of parametric tests that are based on the assumption that samples are drawn from normally, distributed random sample. If the k2nn distances in a set of randomly spatially distributed points are not normally distributed, how can a field of points be measured for randomness? One technique involves using probability theory to estimate a correcting factor that converts a skewed finite field measurement into its corresponding normal distribution[5 at Chap. 18]. Squyres considered non-normality in the context of developing a correction factor for lunar crater counting, [6], and Kreslavsky considered the statistical robustness against non-normality of methods to analyze

the spatial distribution of craters[7].

Another approach to measure for randomness of points in a finite field is to compare the distribution of a simulated random field k2nn distances to the k2nn distances measured from an observation field[3, 5 at p. 350, 8]. Both the data and reference distributions are non-normal, but if the two distributions are sufficiently similar, then the inference can be made that the non-normal observation field is equivalent to the simulated distribution of normally distributed points. One accepted method for statistically testing the similarity between two non-normal distributions is the Kolmogorov-Smirnov test[3]. The sensitivity of the Kolmogorov-Smirnov test complicated by the fact that the test's results are sensitive to the shape of the observation frame. A tall and narrow observation frame will not return the same Kolmogorov-Smirnov test as a same area observing frame that has a short and wide shape, and this distortion of kthnn distances in a finite observing frame is known as the "edge effect"[6-9]. The theory of the Kolmogorov-Smirnov test is described in Bain and Engelhardt's *Introduction to Probability and Mathematical Statistics*[10 at p. 460-461], and a sample KS computation can be found in [11 at Prob. 20-10]. In summary, the Kolmogorov-Smirnov test finds the absolute value of the maximum distance between an empirical reference and an empricial observation distribution, denoted "D" in the test, Figure 5, p. 19. The value of D is characterized, and from a table of KS critical values, the test returns a p-value indicating whether the two distributions are equal. Critical values for the Kolmogorov-Smirnov test are described in Table 2, p.15[1-2].

The Kolmogorov-Smirnov test has been used for comparing two distributions of cumulative size frequency lunar crater counts[9, 12-13], and the test is one technique recommended by the 1979 *Crater Analysis Working Group*[3], a NASA sponsored working group that established the astrogeology community's

standard practice for preparing cumulative spatial frequency diagrams (CSFD) for lunar and planetary bodies. When a crater bin is found to be non-random, the field is examined for secondary impact contamination using lunar geology procedures[14-15], the secondary impact craters are eliminated from CSFD, and then the randomness test of the crater bin is repeated[9].

1.3 Results

Surrounding the four kilometer diameter lunar crater Hell Q at selenographic coordinates latitude 33.0°, longitude 4.4°W, four potential fields were identified for cumulative spatial frequency counting in support of possible absolute modeling age (AMA) dating, Figure 6, p. 20, and 890 craters were preliminarily counted in a 16.4 square kilometer observing frame due north of Hell Q, Figures 6 and 7. The observation field is identified as "M126961088LE North" in Figure 7, and the observing field image is a one square meter resolution orthogonal projection of Lunar Reconnaissance Orbiter (LRO) image M126961088LE.CDR prepared by Fisher using *ISIS* version 3.4 image processing software of the Astrogeology Research Program of the U.S. Geological Survey. The USGS is both the archivist of NASA planetary satellite imagery, and the agency distributes generally accepted software for image processing and analysis of that imagery.

Using standard procedures recommended by [3] and [9], 890 craters were separated into six 2^n bins. The characteristics of the craters in each bin are described in Table 3, p. 15, and graphical plots of the positions of craters in each bin are shown in Figure 8, p. 22. Additionally, simulation was used to generate one-hundred instances of each of six frames that contained the same number of points as their actual counterpart bins and that were the same dimensions as their actual counterpart bins. The points in the simulated frames were spatially distributed randomly using a random number generator. Files containing the k2nn distances in each bin of actual counted craters and the k2nn distances in

the corresponding reference fields were generated. *Mathematica* was used for the foregoing processes, and for replication purposes, supporting computer code is listed in Appendix D.

The actual k2nn distances for five crater diameter bins and the corresponding simulated reference k2nn distances for each crater diameter were compared using the Kolmogorov-Smirnov test. The sixth binned field only contains one point, and that field cannot be evaulated for randomness. The statistical software R was used as the computational engine, and for replication purposes,n 1 is included in Appendix C. Results of the Kolmogorov-Smirnov test for the five bins are reported in Table 1, p. 14 at a significance level $\alpha = 0.10$. As noted in the Introduction above, Figure 1, p. 17, shows a plot of the distribution of observation field of k2nn distances, standardized as empirical continuous distributions, compared to a standardized distribution of k2nn distances in a same dimensioned field created by random number simulation.

Based on that analysis, three of the five crater diameter bins are spatially distributed randmonly, Table 1. The rejection of the two crater diameter bins as not spatially random may be an artifact of R's ks.test function. When performing the Kolmogorov-Smirnov comparision tests, R stated the following: "Warning message: In ks.test(dist1, dist2, alternative ='two.sided') : p-values will be approximate in the presence of ties." The p-values given for the two smallest sized bins (n=14 and n=58) were inconsistent with the critical p-values provided by the *CRC Standard Mathematical Tables and Formulae*[1-2]. R reported a p-value rejection at a D statistic that the CRC tables indicate that R's reject may be related to the number of craters in the bin. Future work will resolve this ambiguity by manually refiguring the KS test for the two rejected bins, using [11] as a computation guide. Confirmation of Kolmogorov-Smirnov test results will also be corroborated by mean 2nd nearest neighbor tests for each diameter bin.

In that regard, a specialty R-package, spatstat[16], will be explored in order to replicate all of the tests done here.

Conversely, the rejection of two crater bins as not spatially random may not be a computation artifact, and their statistical rejection may actually be the result of field contamination by secondary crater impacts. According to Shoemaker[14] and Gault[15], secondary impact craters can be identified principally by their combined composite, elongate, and rounded rim characteristics [14 at p. 32-36][15 at p. 133-134, 139]. Secondary impact craters are the result of swarms of small craters ejected at slow-speeds and at low-angles from a primary impact, and typically secondary impacts are ellipitical and overlap other craters of the ejection swarm, Figure 9, p. 23. Fourteen craters of the preliminary 890 count are ellilpitcal and-or overlapping. Two other small craters have slumped walls indicative of a slow-speed impact. In future work, these craters will be geologically evaluated for exclusion from the inventory as secondary impacts. Removal of secondaries and retesting of the two rejected fields, may result in their Kolmogorov-Smirnov test results changing to spatially random. There is one remaining diameter bin of craters to count in the observing field, and the bin contains an estimated 600 additional craters with diameters between 2 and 4 meters. The decision to continuing counting the field will be evaulated after the rejection of two of the five crater bins is resolved.

For persons interested in reading about spatial analysis theory and practice using R, the following sources are recommended - [5], [8], [17] and [18].

2 Task 2: Map orthogonal coordinates to selenographic latitude and longitude.

2.1 Summary and Conclusion

In order to do corroborative spatial randomness analysis of the 890 craters using a generally accepted lunar crater analysis package, *CraterStats* [9,20,21], pixel x,y points used to mark the location of the observation field craters need to be mapped to high-precision lunar longitude and latitude coordinates. Although exact values at fourteen signficiant digits of latitude and longitude degrees for craters can be manually extracted from the observing frame with U.S. Geological Survey ISIS Qview software, it was hoped that the manual assembly of 890 latitude and longitude values to fourteen signficiant digits could be avoided by linear regression. Although a highly accurate linear regression was found, residuals had a precision of plus or minus 0.00002 degrees, and by inspection of the source lunar image of the field, that precision equates to about plus or minus ten meters. Since that level of precision might affect the results of future spatial randomness analysis, we concluded that the manual labor of extracting exact latitude and longitude values in degrees will have to be expended.

2.2 Methods and Results

In order to uniquely locate an individual crater at image resolutions of one square meter in a selenographic coordinate system based on degrees, a precision of 11 significant digits is required. Traditional approaches for finding high-precision degree positions include plate solving using linear regression implemented by the least squares method from linear algebra [22] and inverse solving trigonmetric orthographic mapping functions [23]. Plate solving is inapplicable to this matter, and for ultra-small lunar surface distances of less than ten meters, the inverse method has its own precision problems resulting

from the inaccuracies of trigonometric functions at very small angles. Therefore, a hybird method of multiple linear regression was tested, in order to see if it might provide a labor saving computational shortcut to finding 890 high precision positions.

A sample consisting of the orthographic x,y pixel coordinates of the four corners of the observing field were combined with thirty-size randomnly selected points from the population of 890 points, Table 4, p. 16. U.S. Geological Survey ISIS Qview software was used to extract the high-precision lunar latitude and longitude positions to twelve significant digits from LRO NAC image M126961088LE[19]. Linear regression modelling was undertaken in order to find best fit models for lunar latitude and longitude based on Table 4.

During regression modeling, the null hypothesis was that there was no combination of transformations of two x,y position variables that was statistically significantly associated to a level $\alpha = 0.00001$ with a lunar latitude or longitude. The alternative hypothesis was that some combination of transformations of two x,y position variables could be statistically significantly associated to a level $\alpha = 0.00001$ with a lunar latitude or longitude. The high α was necessary in order to generate high-precision coordinates in the degrees.

One outlier was excluded from the models, and the following best fit mapping equations were derived after several attempts to optimally minimize least residuals, minimize Akake's Information Criteria, and maximize p-value rejection of the null hypothesis:

$$\begin{aligned}
\text{longitude} = (3.55506773145 \times 10^{2}) \quad & +(3.91987850137 \times 10^{-5}) \times x) \\
& +(-4.64324471343 \times 10^{-9}) \times y). \\
\text{latitude} = (-3.25633152816 \times 10^{1}) \quad & +(0 \times x) \\
& +(-3.28696665477 \times 10^{-5}) \times y) \\
& +(-2.5658939655 \times 10^{-12}) \times y^{2}) \\
& +(1.68934247952 \times 10^{-15}) \times y^{3}).
\end{aligned}$$

The longitude model had normally distributed residuals, Figure 10, p. 23, and the longitude mapping model had a statistically significant goodness of fit (R^2 = 0.99996849522, F-statistic=1174387.9, df(1,37), p-value =< 0.00001). Similarly, the latitude model had normally distributed residuals, Figure 11, and the longitude mapping model had a statistically signficant goodness of fit (R^2 = 0.999999854766, F-statistic=80330363.1, df(3,35), p-value =< 0.00001). Interestingly, the x coordinate was not used in the cubic polynomial regression model for latitude. At such high F-statistic levels, the null hypothesis was rejected and the alternative hypothesis was accepted for both the longitude and latitude models. Following development of the models, the 890 x,y observation points were mapped to lunar latitude and longitudes in degree units to twelve significant digits.

However, on reflection, the precision of the residuals of the method, illustrated in Figures 10 and 11, only extends to at best the fifth significant digit, but precision at the eleventh significant digits is required. That level of precision in estimating degrees corresponds to about ten physical meters on the lunar surface, and as shown Figure 3, p. 18, a significant portion of the 890 crater sample have second nearest neighbor distances of less than fifty meters. Therefore, analysis stopped at that step, and there was no need to construct confidence or prediction intervals for the model. The labor of extracting the eleventh significant digit precision latitudes and longitudes manually cannot be avoided.

In conclusion, two procedures were executed with favorable results in support of an ongoing research project[24,25], and that program has evolved into AMA dating of lunar crater Hell Q. Further work will be needed to determine if two of the six diameter binned classes of craters are spatially distributed random or non-randomly either by manual computation or by using the *spatstat* R package.

References

[1]. Zwillinger, D. (ed.) 2003. (31 Ed.) Table 7.14.9. Critical values, two sample Kolmogorov-Smirnov Test. Section 7.14. In *CRC Standard Mathematical Tables and Formulae*. Boca Raton, La.:Chapman and Hall-CRC.

[2]. – Table 7.14.8. Critical values, Kolmogorov-Smirnov Test. Section 7.14. In *CRC Standard Mathematical Tables and Formulae*. Boca Raton, La.:Chapman and Hall-CRC.

[3]. Crater Analysis Techniques Working Group, Arvidson, R. E., Boyce, J. M., Chapman, C. R., Cintala, M. J., Fulchignoni, M., Moore, H., et al. (1979). Standard techniques for presentation and analysis of crater size-frequency data. *Icarus*, 37(2), 467474. doi:10.1016/0019-1035(79)90009-5.

[4]. Perry, G.L.W., Miller, B. and Enright, N.J. (2006). A Comparison of Methods for the Statistical Analysis of Spatial Point Patterns in Plant Ecology. *Plant Ecology*. 187(1):59-82.

[5]. Gelfand, A. E. (2010). *Handbook of spatial statistics*. Boca Raton: CRC Press.

[6]. Kreslavsky, M. A. 2007. Statistical Characterization of Spatial Distribution of Impact Craters: Implications to Present-Day Cratering Rate on Mars. *Seventh International Conference on Mars*, held July 9-13, 2007 in Pasadena, California. LPI Contributions No. 1353:3325. url: `http://www.lpi.usra.edu/meetings/7thmars2007/pdf/3325.pdf`. Accessed 2013-3-24.

[7]. Squyres, S.W., Howell, C., Liu, M.C. and Lissauer, J.J. 1997. Investigation of Crater "Saturation" Using Spatial Statistics. *Icarus*. 25(1):67-82. Doi: 10.1006/icar.1996.5560.

[8]. Baddelely, A. 2011. *Analyzing spatial point patterns in R.* (Workshop Notes). Perth, Australia:CSIRO and University of Western Australia. Retrieved Apr. 6, 2013, from `http://www.csiro.au/resources/pf16h`.

[9]. Michael, G.G., Platz, T., Kneissl, T., Schmedemann, N. 2012, June. *Planetary surface dating from crater sizefrequency distribution measurements: Spatial randomness and clustering*. Slide Presentation. U.S. Geological Serv. *Planetary Data: A Workshop for Users and Software Developers* held June 25-29, 2012, at Northern Arizona University (NAU), Flagstaff, Arizona. url: `ftp://pdsimage2.wr.usgs.gov/pub/pigpen/tutorials/FreieUni_Workshop2012/5_randomnessAnalysis.pdf`. Accessed 2013-3-24.

[10]. Bain, L. J., & Engelhardt, M. (2009). *Introduction to probability and mathematical statistics.* Belmont, CA: Brooks/Cole Cengage Learning.

[11]. Research and Education Association. (1978). *The statistics problem solver.* New York: REA.

[12]. Fassett, C.I., Head, J.W., and Kadish, S.J., et al. 2012. Stratigraphy, Sequence, and Crater Populations of Lunar Impact Basins from Lunar Orbiter Laser Altimeter (LOLA) Data: Implications for the Late Heavy Bombardment. *Workshop on the Early Solar System Bombardment II*, held 1-3 February 2012, in Houston, Texas. LPI Contribution No. 1649, pp.18-19. url: `http://www.lpi.usra.edu/meetings/bombardment2012/pdf/4023.pdf`.

[13]. Head, J.W., Fassett, C.I., Kadish, S.J., Smith, D.E. et al. 2010. Global Distribution of Large Lunar Craters: Implications for Resurfacing and Impactor Populations. *Science.* 329(5998):1504-1507. Doi: 10.1126/science.1195050.

[14]. Shoemaker, E. (1966). Preliminary Analysis of the Fine Structure of the Lunar Surface of Mare Cognitum. Chap. 2 in Conference on the Nature of the Surface of the Moon, In Hess, W. N., IAU Commission 17: The Moon., United States., & Goddard Space Flight Center. (1966). *The nature of the lunar surface: Proceedings of the 1965 IAU-NASA Symposium. Baltimore: Johns Hopkins Press.*

[15]. Gault, D.E., Quaide, W.L., and Oberbeck, V.R. (1966). Interpreting Range Photographs from Impact Cratering Studies. Chap. 6 in Conference on the Nature of the Surface of the Moon, In Hess, W. N., IAU Commission 17: The Moon., United States., & Goddard Space Flight Center. (1966). *The nature of the lunar surface: Proceedings of the 1965 IAU-NASA Symposium.* Baltimore: Johns Hopkins Press.

[16]. Baddeley, A. and Turner, R. et al. (2013, Mar. 1). *Spatstat, an R Package.* Computer software. (Version 1.31-1). Retrieved Apr. 3, 2013 from `http://cran.r-project.org/web/packages/spatstat/index.html`. *See also* `http://www.spatstat.org/`.

[17]. Diggle, P. (2003). *Statistical analysis of spatial point patterns.* London: Arnold.

[18]. Bivand, R. S., Gómez-Rubio, V., & Pebesma, E. J. (2008). *Applied spatial data analysis with R.* New York: Springer.

[19]. NASA and Arizona State. Univ. 2010, Apr. 26. *Lunar Reconnaissance Orbiter NAC Image M126961088LE.CDR.* Retrieved 2011 from `http://wms.lroc.asu.edu/lroc/view_lroc/LRO-L-LROC-2-EDR-V1.0/M126961088LE`.

[20]. Kneissl T., van Gasselt S., and Neukum G.. 2001. Map-projection-independent crater size-frequency determination in GIS environments New software tool for ArcGIS. *Planetary and Space Science*. 59 (1112):12431254.

[21]. Michael, G. G. & Neukum, G. (2010). Planetary surface dating from crater size-frequency distribution measurements: Partial resurfacing events and statistical age uncertainty. *Earth & Planetary Sci. Ltr.*, 294, pp. 223-229, doi:10.1016/j.epsl.2009.12.041.

[22]. Berry, R., and Burnell, J. (2009). *Handbook of astronomical image processing.* Richmond, Va: Willmann-Bell.

[23]. Snyder, John P. (1987). *Map Projections: A Working Manual.* U.S.G.S. Professional Paper: 1395. Retrieved Jan. 27, 2013 from `http://pubs.er.usgs.gov/publication/pp1395`.

[24]. Fisher, Kurt A. 2012. New Developments in Counting Very Small Craters. *Selenology Today.* pp. 1-24. url: `http://www.lunar-captures.com//Selenology_Today/selenologytoday26.pdf`. Accessed 2013-3-24.

[25]. Fisher, Kurt A., Univ. of Utah. 2012. New Developments in Counting Very Small Craters. *Univ. of Utah. Undergraduate Research Abstracts.* 2012:93. url: `http://content.lib.utah.edu/cdm/compoundobject/collection/UROP/id/2567/rec/5`. Accessed 2013-3-24.

Appendices

A Tables

Table 1: **Kolmogorov-Smirnov Two Sample Test Results of 2nd Nearest Neighbor Distances by Binned Crater Diameters (N=890)** $\alpha = \mathbf{0.10}$

Bin	D_{low}	D_{upp}	n	KS stat.	KS p-value	KS c.v.	Random
1	4	6	347	0.814	0.20	0.087	Y
2	6	8	283	0.0812	0.16	0.096	Y
3	8	12	187	0.0599	0.33	0.118	Y
4	12	20	58	0.091	0.003	0.21	N†
5	20	36	14	0.142	< 0.0001	0.314‡	N†
6	36	68	1	na	na	na	

Note: $\alpha = 0.10$, two-sided. KS stat. = the Kolmogorov-Smirnov test statistic from comparing the distribution of observed field k2nn distances with a simulated random field. The KS statistic usually is denoted as D, but here KS is used in order to avoid confusion with the use of D for diameter. KS p-value = the p-value of corresponding to the KS statistic for a field comparision. KS c.v = the KS critical value at the α significance level per Table 2. Random = Conclusion of whether bin-field is spatially random. Y=Yes; N=No. † = R's ks.test function returned a rejection p-value inconsistent with the critical value in Table 2 and [1-2]. If the critical values used in Table 2 and [2] were used, these fields would be random. *See* discussion in main text. ‡ - critical values for $n <= 40$, see [2]. All distances are in meters. D = min. Feret's diameter of crater.

Table 2: **Kolmogorov-Smirnov Two Sample Test Critical Values**

$H_0 : d_1 = d2; H_a : d_1 \neq d2.$

Reject H_0; accept H_a, if $KS >=$ critical value.

$KS = \max |F_{n_1}(x) - F_{n_2}(x)|$

For $n > 40$:

p-value	Critical value of KS
$\alpha = 0.20$	$KS >= 1.07\sqrt{\frac{n_1+n_2}{n_1 n_2}}$.
$\alpha = 0.10$	$KS >= 1.22\sqrt{\frac{n_1+n_2}{n_1 n_2}}$.
$\alpha = 0.05$	$KS >= 1.36\sqrt{\frac{n_1+n_2}{n_1 n_2}}$.
$\alpha = 0.01$	$KS >= 1.63\sqrt{\frac{n_1+n_2}{n_1 n_2}}$.
$\alpha = 0.001$	$KS >= 1.96\sqrt{\frac{n_1+n_2}{n_1 n_2}}$.

Note: Source: [1]. The KS statistic usually is denoted as D, but here KS is used in order to avoid confusion with the use of D for diameter. For critical values for $n <= 40$, see [2].

Table 3: **Binned Crater Counts with 2nd Nearest Neighbor Distance Characteristics (N=890)**

D_{low}	D_{upp}	n	μ k2nnd	σ^2 k2nnd	σ k2nnd
4	6	347	114.8	3879.0	62.3
6	8	283	122.4	5171.6	71.9
8	12	187	163.4	7817.6	88.4
12	20	58	293.1	28720.2	169.45
20	36	14	646.0	110674.0	332.7
36	68	1	na	na	na

Note: All distances are in meters. D = min. Feret's diameter of crater. The bins are open at the top of the interval, that is for bin 4-6, the diameters of craters in the bin are greater than or equal to 4 meters and less than 6 meters.

Table 4: **Orthographic and high-precision selenographic coordinates for 40 points in the observing field.**

x	y	Long.◦	Lat.◦
26	2000	355.507811334742	−32.629129380662
2216	2000	355.593610505426	−32.629132251689
81	9500	355.509810090783	−32.876432892440
2270	9500	355.595847823001	−32.876434307129
456	2093	355.524658639767	−32.632234863937
568	2152	355.529047307384	−32.634143704559
1479	2619	355.564729859673	−32.649528053623
1064	2632	355.548444092958	−32.650005238215
830	2916	355.539379067319	−32.659305208823
873	3131	355.540935886703	−32.666459822400
1152	3321	355.551983752754	−32.672661069306
832	3326	355.539376708248	−32.672898620385
515	3731	355.526907140286	−32.686251359858
1322	3838	355.558642100542	−32.689712402966
1428	3946	355.562751099896	−32.693289415467
236	4025	355.515994090206	−32.695906888647
803	4146	355.538238445718	−32.699965996756
1714	4475	355.573947914593	−32.710816487511
326	4499	355.519530116917	−32.711528377075
1405	4752	355.561903083981	−32.719879972293
2034	5022	355.586566702806	−32.728819132168
635	5116	355.531712172181	−32.731921230892
295	5178	355.518387234885	−32.733945122421
1688	5484	355.572112081581	−32.744084629859
247	5521	355.516397855922	−32.745272284387
1190	5563	355.553399035859	−32.746709077347
216	5593	355.515262658717	−32.747656719460
1835	5684	355.578776803088	−32.750641704734
2062	5716	355.587567284311	−32.751712869891
1266	5827	355.556376337586	−32.755413556365
808	5909	355.538369385254	−32.758035740072
1620	6511	355.570274985987	−32.777948842223
198	6572	355.514539162653	−32.779970343530
850	6685	355.540066565695	−32.783672389972
902	7682	355.542047408981	−32.816582566205
1176	8142	355.552829993674	−32.831726532989
1238	8156	355.555242379194	−32.832203504256
2240	8801	355.594701678258	−32.853421646604
650	8988	355.540053888582	−32.859627507468
780	3661	355.537356001242	−32.683928079998

Note: The first four points are the corners of the observing frame shown in Figures 6 and 7, and the remaining craters were randomnly selected from 890 observed craters. U.S.G.S. ISIS Qview software was used to extract high-precision lunar latitude and longitude positions to twelve significant digits from LRO NAC image M126961088LE[19], and that level of precision is needed to uniquely locate 1 meter points using the selenographic degree-based coordinate system.

B Figures

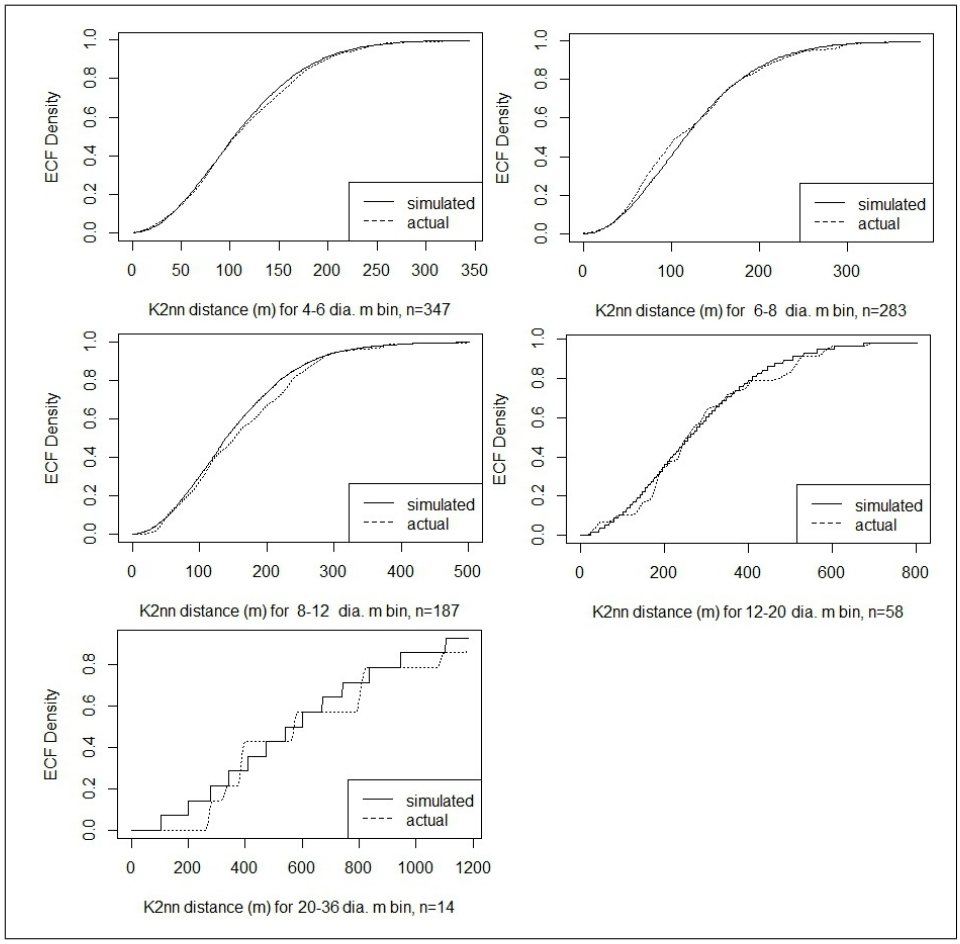


Figure 1: Cumulative Empirical Continuous Function Distributions of actual and simulated fields.

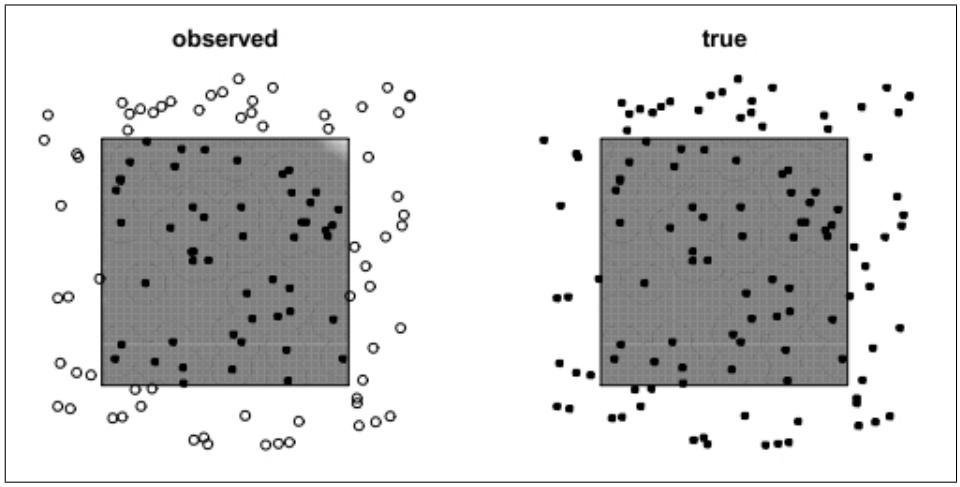


Figure 2: Information loss about the true k2nn distance of a point caused by the imposition of a finite observing frame on an infiinte spatially random field. Figure adapted from Baddelely[8] at p. 117.

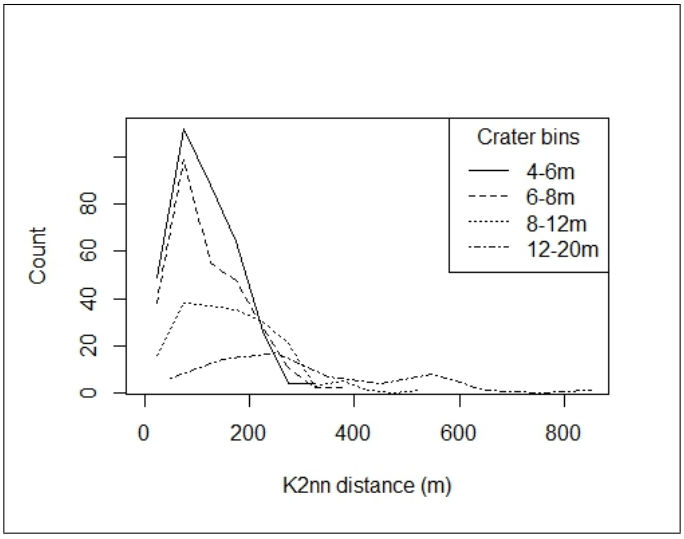


Figure 3: Histogram distributions of k2nn crater distances in 4 of 5 actual fields.

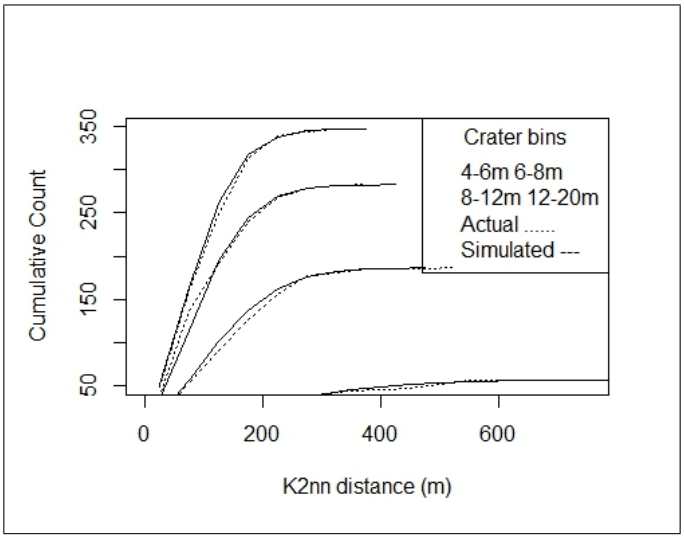


Figure 4: Cumulative 2nd nearest neighbor distance distributions of 4 of 5 actual and simulated fields.

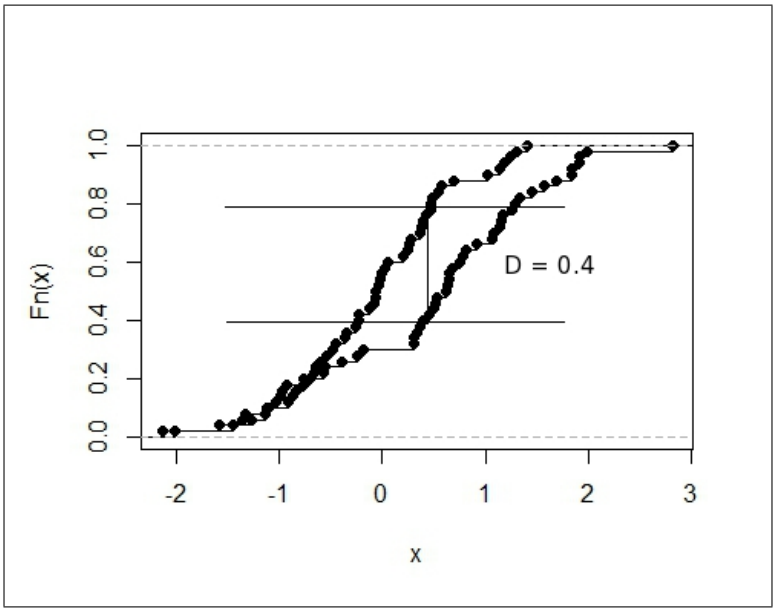


Figure 5: Illustration of two empirical CDFs compared using the Kolmogorov-Smirnov test. After Bain[10], Figure 13.1. D is the critical characteristic, and D is the maximum y-distance between the two ecdf's.

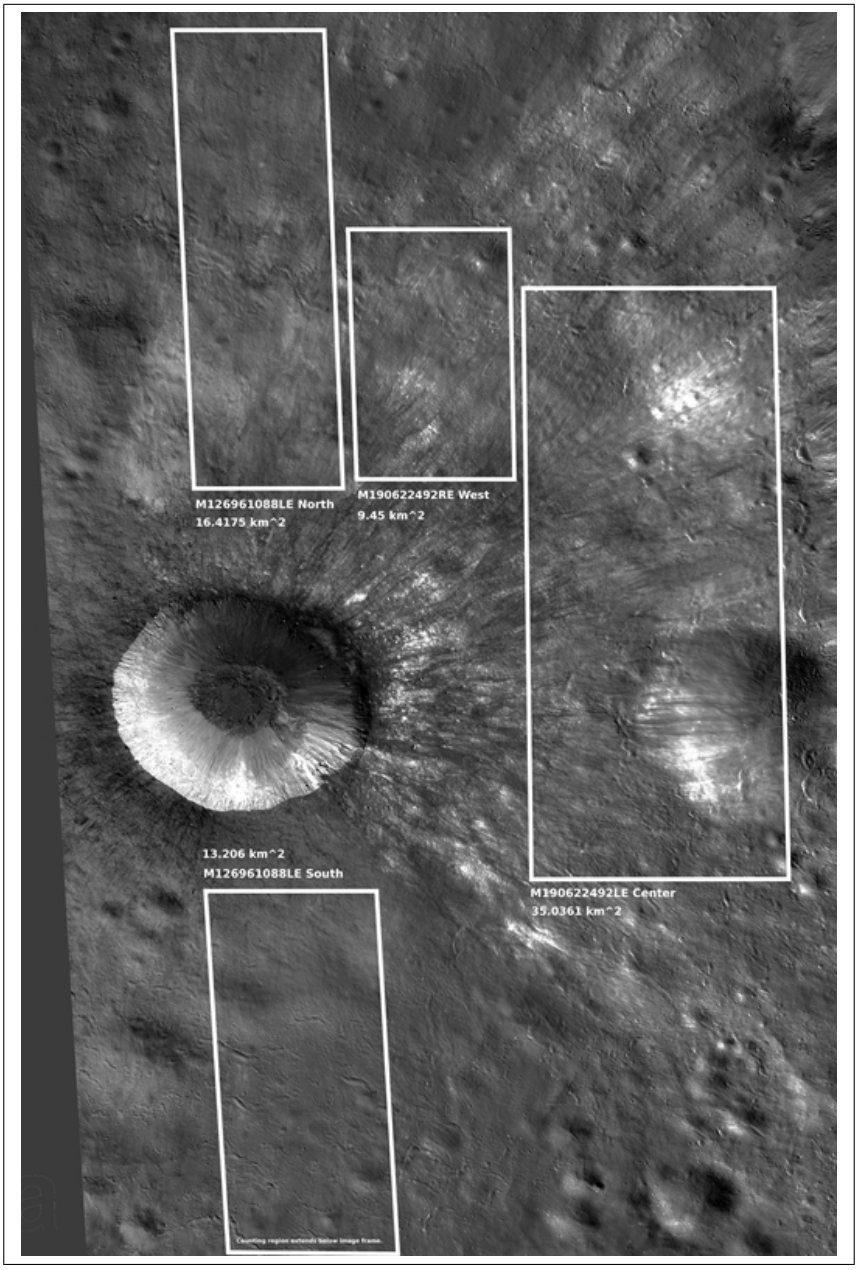


Figure 6: Overview of the area surrounding lunar crater Hell Q made from a mosaic of three LRO Narrow Angle Camera images including LRO image M126961088LE. North is up; lunar west is left. The source image resolution is one square meter per pixel. The crater counting field discussed in this paper is marked by the white parallelogram due north of Hell Q. Scale: Hell Q has a 4 km dia.
Note: *See* `http:\\lpod.wikispaces.com/November+7,+2012` for a wider view.

Figure 7: An expanded view of the crater counting field with proportionally sized dots marking some of the 890 craters counted in the field. Scale: The field is approx. 2.2 km wide.

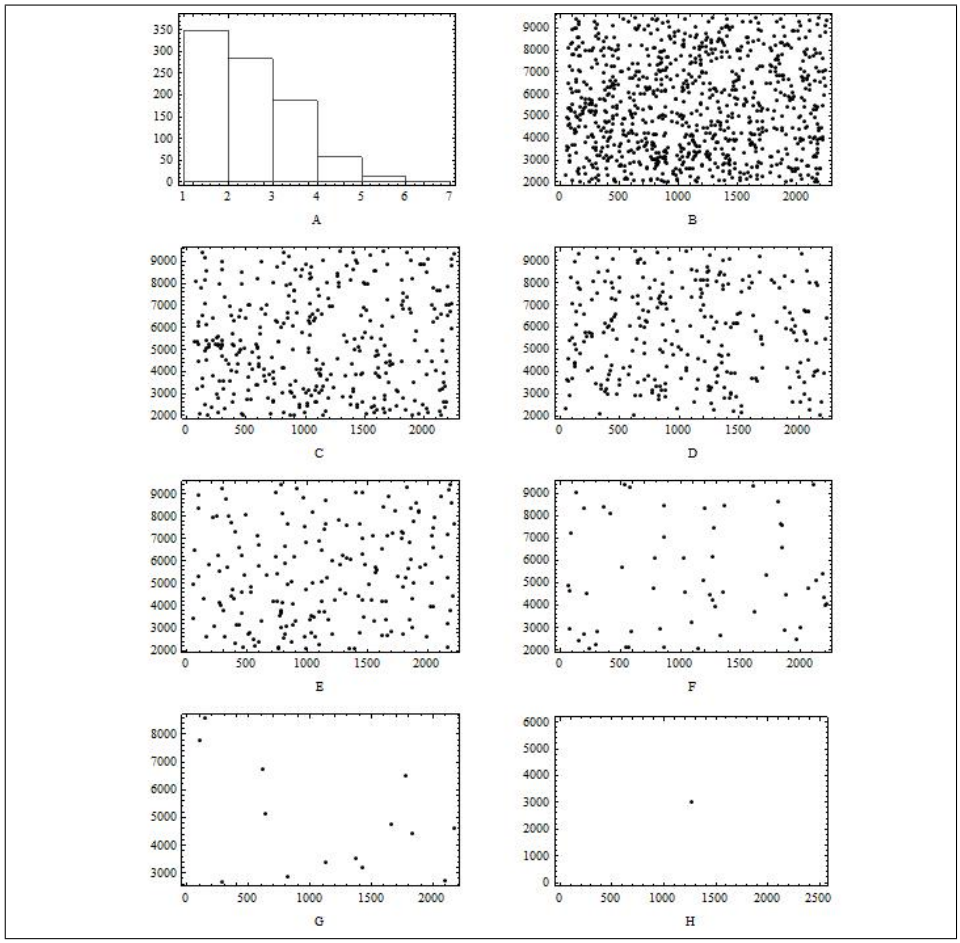


Figure 8: A histogram of the crater counts in each bin, position plots for craters for the entire field, and position plots for craters for craters in each of the six diameter bins.
Note: (a) A histogram of the crater counts in each bin. (b) Plot of the positions of all 890 craters in observing field. (c) Plot of the positions of 347 craters in diameter bin 4-6 meters. (d) Plot of the positions of 283 craters in diameter bin 6-8 meters. (e) Plot of the positions of 187 craters in diameter bin 8-12 meters. (f) Plot of the positions of 58 craters in diameter bin 12-20 meters. (g) Plot of the positions of 14 craters in diameter bin 20-36 meters. (h) Plot of the position of one crater in diameter bin 36-68 meters. A computer generated y-axis artificially makes the one crater in 'h" appear at the center of the frame. Clustering of craters can be seen in (c), (d), (e) and (f), but is this clustering real or imagined? Are the craters in each field spatially distributed randomly?

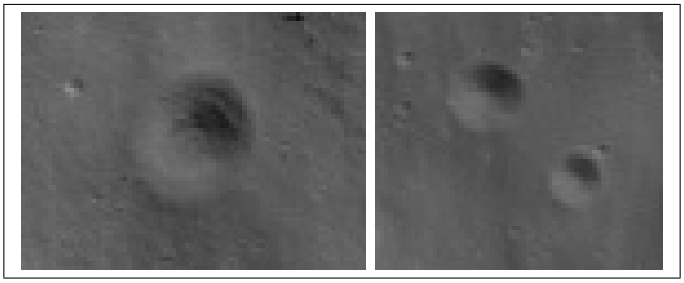


Figure 9: Left: A twenty-four meter diameter dia. secondary impact crater from the sample. This crater's elliptical shape, its rounded rim, and sluffing of at the northeast side wall indicate that this crater is probably a slow speed secondary impact. Right: A twelve and eight meter crater from the observation field that are primary impacts. The upper left crater has a sharp conical and circular shape. Unlike the left ellpitical crater, both of these craters have well-defined linear shadows across the bottom of the crater, and that shadow shape indicates the crater's are circular. *Compare* that shadow shape to the ill-defined, rim shadow for the secondary impact crater on the left.

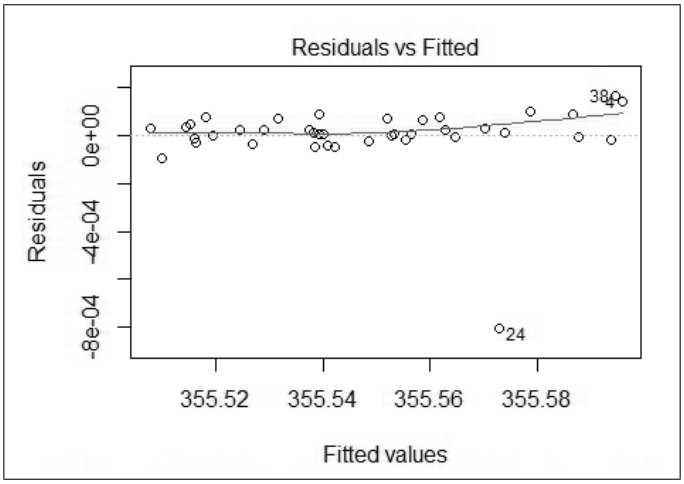


Figure 10: Residuals of best fit model for mapping $x + y \rightarrow$ lunar longitude.

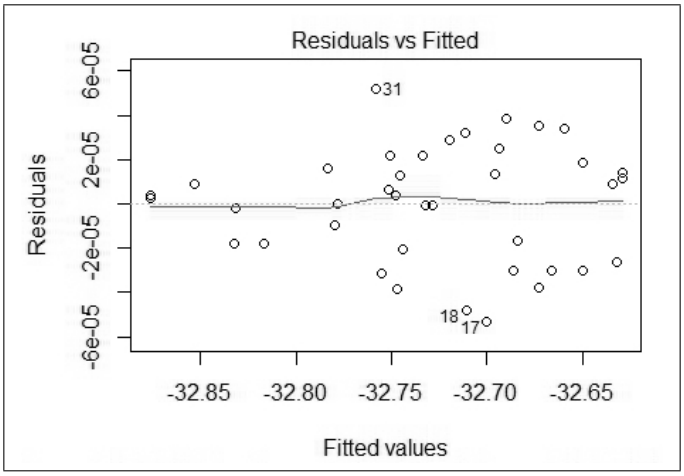


Figure 11: Residuals of best fit model for mapping $y + y^2 + y^3 \rightarrow$ lunar latitude.

C - Code for Kolmogorov-Smirnov Test Analysis in R.

```
# Stat 3080, R Group Spatial Randomness Crater Project
# Kurt Fisher and Russell Fuller
# Task 1 - Plot simulated bins
# Version Date: 4-19-2013
# Version: 1.3
# File: 20130406PlotSimBins.R

# Uses: tblKnnBinX.txt

# Makes the following figures and tables that will be used in writing the report.
# Make manually:
#    A set of line histogram and culumlative plots for the kthnn distances of
#    lunar craters. All plots should be save manually.

# Set directory
#   Usage Note:
#   Change this path to your personal working directory.
#   Put data files schooldatamodASCII.txt and schoolacronyms.txt into that directory for reading.
setwd("C:\\Users\\fisherka\\Documents\\00000000UofUSpr2013\\3080Stats\\ProjectPri\\Reports\\20130404Report\\
plotSimBins")

# 1) Get bin data simulated
# a) Get data Bin1 simulated
simbin1 = read.table("tblNSimsKthnnBin1.txt", header=TRUE)
simbin1$kthnnDistance = sort(simbin1$kthnnDistance)
simbin1 <- data.frame(simbin1)
names(simbin1)
# length(simbin1$kthnnDistance)
# plot(sort(simbin1$kthnnDistance))

# b) Get data Bin2 simulated
simbin2 = read.table("tblNSimsKthnnBin2.txt", header=TRUE)
simbin2$kthnnDistance = sort(simbin2$kthnnDistance)
simbin2 <- data.frame(simbin2)
# length(simbin2$kthnnDistance)
# plot(sort(simbin2$kthnnDistance))

# c) Get data Bin3 simulated
simbin3 = read.table("tblNSimsKthnnBin3.txt", header=TRUE)
simbin3$kthnnDistance = sort(simbin3$kthnnDistance)
simbin3 <- data.frame(simbin3)
# length(simbin3$kthnnDistance)
# plot(sort(simbin3$kthnnDistance))

# d) Get data Bin4 simulated
simbin4 = read.table("tblNSimsKthnnBin4.txt", header=TRUE)
simbin4$kthnnDistance = sort(simbin4$kthnnDistance)
simbin4 <- data.frame(simbin4)
# length(simbin4$kthnnDistance)
# plot(sort(simbin4$kthnnDistance))

# e) Get data Bin5 simulated
simbin5 = read.table("tblNSimsKthnnBin5.txt", header=TRUE)
simbin5$kthnnDistance = sort(simbin5$kthnnDistance)
simbin5 <- data.frame(simbin5)
# length(simbin5$kthnnDistance)
# plot(sort(simbin5$kthnnDistance))

# 2) Build histogram data simulated
```

```
# hist(simbin1$kthnnDistance) # look at the histogram
simhistbin1 <- hist(simbin1$kthnnDistance, plot=FALSE) # get the histogram plot values
# hist(simbin2$kthnnDistance) # look at the histogram
simhistbin2 <- hist(simbin2$kthnnDistance, plot=FALSE) # get the histogram plot values
# hist(simbin3$kthnnDistance) # look at the histogram
simhistbin3 <- hist(simbin3$kthnnDistance, plot=FALSE) # get the histogram plot values
# hist(simbin4$kthnnDistance) # look at the histogram
simhistbin4 <- hist(simbin4$kthnnDistance, plot=FALSE) # get the histogram plot values
# hist(simbin5$kthnnDistance) # look at the histogram
simhistbin5 <- hist(simbin5$kthnnDistance, plot=FALSE) # get the histogram plot values

# 3) Get bin data actual
# a) Get data Bin1 actual
actbin1 = read.table("tblKthnnBin1.txt", header=TRUE)
actbin1 <- subset(actbin1, select="kthnnDistance")
actbin1$kthnnDistance <- sort(actbin1$kthnnDistance)
actbin1 <- data.frame(actbin1)
names(actbin1)
length(actbin1$kthnnDistance)
plot(actbin1$kthnnDistance)

# a) Get data Bin2 actual
actbin2 = read.table("tblKthnnBin2.txt", header=TRUE)
actbin2 <- subset(actbin2, select="kthnnDistance")
actbin2$kthnnDistance <- sort(actbin2$kthnnDistance)
actbin2 <- data.frame(actbin2)
names(actbin2)
length(actbin2$kthnnDistance)
plot(actbin2$kthnnDistance)

# a) Get data Bin3 actual
actbin3 = read.table("tblKthnnBin3.txt", header=TRUE)
actbin3 <- subset(actbin3, select="kthnnDistance")
actbin3$kthnnDistance <- sort(actbin3$kthnnDistance)
actbin3 <- data.frame(actbin3)
names(actbin3)
length(actbin3$kthnnDistance)
plot(actbin3$kthnnDistance)

# a) Get data Bin4 actual
actbin4 = read.table("tblKthnnBin4.txt", header=TRUE)
actbin4 <- subset(actbin4, select="kthnnDistance")
actbin4$kthnnDistance <- sort(actbin4$kthnnDistance)
actbin4 <- data.frame(actbin4)
names(actbin4)
length(actbin4$kthnnDistance)
plot(actbin4$kthnnDistance)

# a) Get data Bin5 actual
actbin5 = read.table("tblKthnnBin5.txt", header=TRUE)
actbin5 <- subset(actbin5, select="kthnnDistance")
actbin5$kthnnDistance <- sort(actbin5$kthnnDistance)
actbin5 <- data.frame(actbin5)
names(actbin5)
length(actbin5$kthnnDistance)
plot(actbin5$kthnnDistance)

# 4) Build histogram data actual
# hist(actbin1$kthnnDistance) # look at the histogram
acthistbin1 <- hist(actbin1$kthnnDistance, plot=FALSE) # get the histogram plot values
# hist(actbin2$kthnnDistance) # look at the histogram
acthistbin2 <- hist(actbin2$kthnnDistance, plot=FALSE) # get the histogram plot values
# hist(actbin3$kthnnDistance) # look at the histogram
```

```
acthistbin3 <- hist(actbin3$kthnnDistance, plot=FALSE) # get the histogram plot values
# hist(actbin4$kthnnDistance) # look at the histogram
acthistbin4 <- hist(actbin4$kthnnDistance, plot=FALSE) # get the histogram plot values
# hist(actbin5$kthnnDistance) # look at the histogram
acthistbin5 <- hist(actbin5$kthnnDistance, plot=FALSE) # get the histogram plot values

# 5) For actual counts, plot all the histograms of counts for 4 of 5 bins on one graph - actual
plot(acthistbin1$mids,acthistbin1$counts, type="l", xlim=c(0,850), xlab="K2nn distance (m)", ylab="Count")
legend("topright", legend = c("4-6m","6-8m","8-12m","12-20m"), lty=1:4, title = "Crater bins")
lines(acthistbin2$mids,acthistbin2$counts, lty=2)
lines(acthistbin3$mids,acthistbin3$counts, lty=3)
lines(acthistbin4$mids,acthistbin4$counts, lty=4)
# lines(acthistbin5$mids,acthistbin5$counts) # Fifth bin omitted for plotting clarity

# 6) Plot the simulated and actual bin counts in one plot - cumulative functions
plot(simhistbin1$mids,cumsum(simhistbin1$counts), type="l", xlim=c(0,755), xlab="K2nn distance (m)",
ylab="Cumulative Count")
legend("topright", legend = c("4-6m 6-8m","8-12m 12-20m","Actual .....","Simulated ---"), title = "Crater bins")
lines(acthistbin1$mids,cumsum(acthistbin1$counts), lty="dotted") # actual bin
lines(simhistbin2$mids,cumsum(simhistbin2$counts), type="l") # simulated bin
lines(acthistbin2$mids,cumsum(acthistbin2$counts), lty="dotted") # actual bin
lines(simhistbin3$mids,cumsum(simhistbin3$counts), type="l") # simulated bin
lines(acthistbin3$mids,cumsum(acthistbin3$counts), lty="dotted") # actual bin
lines(simhistbin4$mids,cumsum(simhistbin4$counts), type="l") # simulated bin
lines(acthistbin4$mids,cumsum(acthistbin4$counts), lty="dotted") # actual bin

# plot(simhistbin5$mids,simhistbin5$counts, type="l", ylim=c(0,8)) # simulated bin
# lines(acthistbin5$mids,acthistbin5$counts, lty="dotted") # actual bin

# 11) Do Ks test and plot the comparision of distributions and plot CDF distributions

# Bin 1
length(simbin1$kthnnDistance)
length(actbin1$kthnnDistance)
a1 <- ecdf(simbin1$kthnnDistance)
b1 <- ecdf(actbin1$kthnnDistance)
col11 <- seq(1,ceiling(max(actbin1$kthnnDistance)),1)
col21 <- a1(seq(1,ceiling(max(actbin1$kthnnDistance)),1))
plot(col11,col21, type="l", xlim=c(0,ceiling(max(actbin1$kthnnDistance))), xlab="K2nn distance (m) for 4-6 dia. m
bin, n=347", ylab="ECF Density")
lines(ksmooth(col11,b1(col11),"normal",bandwidth=15), lty="dotted")
legend("bottomright", legend = c("simulated","actual"), lty=1:2)
ks.test(a1(col11),b1(col11), alternative="two.sided")

# Bin 2
length(simbin2$kthnnDistance)
length(actbin2$kthnnDistance)
a2 <- ecdf(simbin2$kthnnDistance)
b2 <- ecdf(actbin2$kthnnDistance)
col12 <- seq(1,ceiling(max(actbin2$kthnnDistance)),1)
col22 <- a2(seq(1,ceiling(max(actbin2$kthnnDistance)),1))
plot(col12,col22, type="l", xlim=c(0,ceiling(max(actbin2$kthnnDistance))), xlab="K2nn distance (m) for 6-8 dia. m
bin, n=283", ylab="ECF Density")
lines(ksmooth(col12,b2(col12),"normal",bandwidth=15), lty="dotted")
legend("bottomright", legend = c("simulated","actual"), lty=1:2)
ks.test(a2(col12),b2(col12), alternative="two.sided")

# Bin 3
length(simbin3$kthnnDistance)
length(actbin3$kthnnDistance)
a3 <- ecdf(simbin3$kthnnDistance)
b3 <- ecdf(actbin3$kthnnDistance)
col13 <- seq(1,ceiling(max(actbin3$kthnnDistance)),1)
```

```
col23 <- a3(seq(1,ceiling(max(actbin3$kthnnDistance)),1))
plot(col13,col23, type="l", xlim=c(0,ceiling(max(actbin3$kthnnDistance))), xlab="K2nn distance (m) for 8-12 dia. m
bin, n=187", ylab="ECF Density")
lines(ksmooth(col13,b3(col13),"normal",bandwidth=15), lty="dotted")
legend("bottomright", legend = c("simulated","actual"), lty=1:2)
ks.test(a3(col13),b3(col13), alternative="two.sided")

# Bin 4
# P-value result is inconsistent with critical value table.
length(simbin4$kthnnDistance)
length(actbin4$kthnnDistance)
a4 <- ecdf(simbin4$kthnnDistance)
b4 <- ecdf(actbin4$kthnnDistance)
col14 <- seq(1,ceiling(max(actbin4$kthnnDistance)),1)
col24 <- a4(seq(1,ceiling(max(actbin4$kthnnDistance)),1))
plot(col14,col24, type="l", xlim=c(0,ceiling(max(actbin4$kthnnDistance))), xlab="K2nn distance (m) for 12-20 dia. m
bin, n=58", ylab="ECF Density")
lines(ksmooth(col14,b4(col14),"normal",bandwidth=15), lty="dotted")
legend("bottomright", legend = c("simulated","actual"), lty=1:2)
ks.test(a4(col14),b4(col14), alternative="two.sided")

# Bin 5
# P-value result is inconsistent with critical value table.
length(simbin5$kthnnDistance)
length(actbin5$kthnnDistance)
a5 <- ecdf(simbin5$kthnnDistance)
b5 <- ecdf(actbin5$kthnnDistance)
col15 <- seq(1,ceiling(max(actbin5$kthnnDistance)),1)
col25 <- a5(seq(1,ceiling(max(actbin5$kthnnDistance)),1))
plot(col15,col25, type="l", xlim=c(0,ceiling(max(actbin5$kthnnDistance))), xlab="K2nn distance (m) for 20-36 dia. m
bin, n=14", ylab="ECF Density")
lines(ksmooth(col15,b5(col15),"normal",bandwidth=15), lty="dotted")
legend("bottomright", legend = c("simulated","actual"), lty=1:2)
ks.test(a5(col15),b5(col15), alternative="two.sided")

# Make a plot that replaces Bain Figure 13.1.
# test if x is stochastically larger than x2
# using R Help example for ks.test
x <- rnorm(50)
x2 <- rnorm(50, .25)
plot(ecdf(x), xlim=range(c(x, x2)), main="")
plot(ecdf(x2), add=TRUE)
plot.new()
ks.test(x, x2, alternative="two.sided")

#  Other special plots
# 5) For actual counts, plot just bin 1  - actual
plot(acthistbin1$mids,acthistbin1$counts, type="l", xlim=c(0,350), xlab="K2nn distance (m)", ylab="Count")
legend("topright", legend = c("4-6m"), lty=1, title = "Crater bin")
# lines(acthistbin2$mids,acthistbin2$counts, lty=2)
# lines(acthistbin3$mids,acthistbin3$counts, lty=3)
# lines(acthistbin4$mids,acthistbin4$counts, lty=4)
# lines(acthistbin5$mids,acthistbin5$counts) # Fifth bin omitted for plotting clarity

#  Other special plots
# n) Compare 347 simulated and actual 2knnd - histogram
plot(simhistbin1$mids,simhistbin1$counts, type="l", xlim=c(0,350), xlab="K2nn distance (m) for 4-6 dia. m bin,
n=347", ylab="Count")
legend("topright", legend = c("4-6m simulated","4-6m actual"), lty=1, title = "Crater bin")
lines(acthistbin1$mids,acthistbin1$counts, lty="dotted") # actual bin

#  Other special plots
# n) Compare 347 simulated and actual 2knnd - cumulative density
```

```
plot(simhistbin1$mids,cumsum(simhistbin1$counts), type="l", xlim=c(0,350), xlab="K2nn distance (m) for 4-6 dia. m
bin, n=347", ylab="Count")
legend("bottomright", legend = c("4-6m simulated","4-6m actual"), lty=1, title = "Crater bin")
lines(acthistbin1$mids,cumsum(acthistbin1$counts), lty="dotted") # actual bin
```

[End]

D - Code for Crater Binning and Kthnn Distance Computation in Mathematica.

Knn for lunar crater randomness analysis

Analyze craters positions for randomness using knn.

Versions

v1 .3 4 - 5 - 2013 -
Corrects error in Kolmogorov - Smirnov test. KS test requires conversion of simulated and observed distributions into empircial continuous distribution before use by KS test.

Uses

A working directory name defined in a SetDirectory statement, below.

globalKnnTimes = an integer, set below, that specifies the number of nearest neighbor for which distance is measured, e.g. -
1 for the first nearest neighbor,
2 for the second nearest neighbor. Default = 2.
k for the kth nearest neighbor.

globalSimulationAveMethodTimes = the number of times simulated bins of random events (crater impacts) are
averaged in order to create a baseline random distribution for an observing frame.
Default = 100.

globalSimulationMeanMethodTimes = the number of times simulated bins of random events are
evaluated for a mean in support of Mean 2nd Closest Neighbor Distance (M2CND) method.
Default = 1000. Maximum is 4000

globalAlphaSig = 0.10 - Signficance criteria for Kolmogorov-Smirnov test. Default is 0.10.

Craters.csv - A csv file of crater data.

Fields in header row1 per ImageJ are:
CraterNo,
InventoryXCoord,
InventoryYCoord,
X,Y,XM,YM,
Major,Minor,Angle,
Feret,FeretX,FeretY,FeretAngle,
MinFeret

CraterBoxDim.txt - A tab delimited dictionary style file of the boundaries of the observation frame.
The file consists of labeled x,y pairs that mark the corners of a parallelogram box.
An example file is:

LowerLeftX	25
LowerLeftY	2000
LowerRightX	2217
LowerRightY	2000
UpperLeftX	81
UpperLeftY	9500
UpperRightX	2270
UpperRightY	9500

For images, pixels plot starting at coordinate (1,1) in the upper left hand corner.
Therefore, the normal cartesian plot of x,y pixels is reflected. The x,y coordinates in CraterBoxDim.txt are similarly reflected to correspond to the cartesian plots produced here. .

Makes

FigCompareDistributionsKS.jpg - This figure is the primary analysis return of this application. The distributions
of the actual and standard baseline distribution are ploted in a combined histogram, and the p-value of the
Kolmogorov-Smirnov test, the bin intervals, and the bin count are printed to the right of each bin's comparision histogram.

This figure answers the question of the whether the pairwise points in the observing frame for each bin are randomnly distributed.

FigCompareDistributionsM2CND.jpg - This figure is the secondary analysis return of this application, and this return replicates the M2CND method used in CraterStats and CraterTools software developed by the Institute of Geosciences, Berlin Free University.

This figure and the M2CND method is replicated in order to compare the M2CND test results to the Kolmogorov-Smirnov test results.

FigBinsActual.jpg - A panel of distribution plots of crater positions classified by 2n crater diameter bins.

FigBinsActual.svg - Same.
tblKthnnBin1.txt - A five column tab separated table of craters in 2n Bin 1.
tblKthnnBin2.txt - A five column tab separated table of craters in 2n Bin 2.
tblKthnnBin3.txt - A five column tab separated table of craters in 2n Bin 3.
tblKthnnBin4.txt - A five column tab separated table of craters in 2n Bin 4.
tblKthnnBin5.txt - A five column tab separated table of craters in 2n Bin 5.
tblKthnnBin6.txt - A five column tab separated table of craters in 2n Bin 6.
Fields in header row of KnnBinN.txt files are:
obsId - Observation id
x - coordinate X
y - coordinate Y
dia - diameter of crater
bin - 2n bin number
kthnnDistance - distance to Kth nearest neighbor.
tblBinDivisions.txt - A list of the bin divisions in n to < n+1 intervals.
tblBinSummary.txt and .Tex - A summary table of the crater bin characteristics including:
LowBinInterval - Bottom (closed) of the bin interval
UpperBInterval - Top (open) of the bin interval (bin is less than this value)
binCount - Number of items (craters) in the bin
binMeanKthnn - Mean kth nearest neighbor distances in the bin
binVar2Ktnn - Variance of the kth nearest neighbor distances in the bin
binSDKthnn - Standard deviation of the kth nearest neighbor distances in the bin
When the bin size is 1, the variance and standard deviaion cannot be computed,
so the placeholder values of Var=0 and SD=0 are returned.

FigSimulateOneBin.jpg and FigSimulateOneBin.svg - Simulation of one bin of same sizes as actual data.
FigNormalitySimulatedBins.jpg and FigNormalitySimulatedBins.svg - Simulation of many bins of same sizes as actual data. This figure shows
the standard baseline distribution used in the subsequent Kolmogorov-Smirnov test comparing the distribution of the actual bins.

tblBinMkthCDSummary - This table summarizes the replication of the M2CND method used in CraterStats and CraterTools software developed by the Institute of Geosciences, Berlin Free University.

Initialize

Globals

```
globalKnnTimes = 2;
(* the number of nearest neighbor for which distance is measured *)

globalSimulationAveMethodTimes = 100;
(* the number of times simulated bins of random events (crater impacts) are
      averaged in order to create a
 baseline distribution of an observing frame. *)
```

```
globalSimulationMeanMethodTimes = 3000; (* the number of times
 the means of simulated bins of random events (crater impacts) are
      gathered in order to use the M2CND method on an observing frame. *)

globalAlphaSig = 0.15; (* Significance level for Kolmogorov-Smirnov test. *)
```

Working directory.

```
(* Setting environment - directories *)

SetDirectory[
  "C:\\Users\\fisherka\\Documents\\00000000UofUSpr2013\\3080Stats\\ProjectPri"];

NotebookDirectory[]
```

```
C:\Users\fisherka\Documents\00000000UofUSpr2013\3080Stats\ProjectPri\
```

```
Directory[]
```

```
C:\Users\fisherka\Documents\00000000UofUSpr2013\3080Stats\ProjectPri
```

```
FileNames[]
```

```
{20130323knnchangenotes.txt, appKnnv13.nb, apps, Articles, Correspondence,
 CraterBoxDim.txt, Craters.csv, crossref.pdf, dataruns, drafts, Dvalues.xlsx,
 FigBinsActual.jpg, FigBinsActual.svg, FigCompareDistributionsKS.jpg,
 FigCompareDistributionsKS.svg, FigNormalityActualBins.jpg,
 FigNormalityActualBins.svg, FigNormalitySimulatedBins.jpg,
 FigNormalitySimulatedBins.svg, FigSimulateOneBin.jpg, FigSimulateOneBin.svg,
 getstart.pdf, knnBinDivisions.txt, knnBinSummary.tex, knnBinSummary.txt,
 Reports, spatial_R_1_09S.pdf, tblKthnnBin1.txt, tblKthnnBin2.txt,
 tblKthnnBin3.txt, tblKthnnBin4.txt, tblKthnnBin5.txt, tblKthnnBin6.txt,
 tblNSimsKthnnBin1.txt, tblNSimsKthnnBin2.txt, tblNSimsKthnnBin3.txt,
 tblNSimsKthnnBin4.txt, tblNSimsKthnnBin5.txt, TemplateForMemo.docx}
```

Functions

```
Get2NBinNumber[{diameterIdx_, minCraterDia_}] := Module[
  {binsTotal, integerBin},
  (* diameterIdx = crater diameter ;
  minCraterDia = minimum crater diameter in dataset ;
  returns bin numbers 1 through Log2 maximum crater diameter
   size.  Note array declaration for input must be used,
  i.e. - Get2NBinNumber[{a,b}], or runtime error occurs in returned value. *)
  integerBin = Ceiling[Log[2, Max[{diameterIdx - minCraterDia, 2}]]]
  (* Max[*,2] forces bin number 1. *)
 ]

UpdateTableKthnn[tblCratersTemp_, globalKnnTimesTemp_] := Module[
  {i, k, thisTableLength, thisPointKthNearestDistanceTemp,
```

```
 thisPointKthNearestDistance, craterPointsBinTemp,
 craterPointsBinTempOut, lstKthnnDistance, col6},
(* tblCratersTemp_ = For one set of bin number categories,
a five column table of crater data containing index,
Xcoord, Ycoord, Diameter, BinNumber;
returns table updated with column 6 – the distance to the 2th nearest neighbor
  for a single bin number. globalKnnTimesTemp concerns whether the 1st,
2nd or nth nearest neighbor is found, and it is set as a global variable.  *)
n = globalKnnTimesTemp; (* search the nth nearest neighbor *)
craterPointsBinTemp =
 Transpose[{tblCratersTemp[[All, 2]], tblCratersTemp[[All, 3]]}];
(* Get the point XY coords *)
thisTableLength = Length[craterPointsBinTemp];
(* Get number of point coordinate pairs *)
(* Handle runtime error for tables of length 1 or
 2 where there is no second Knn distance *)
If[thisTableLength == 1,
 col6 = {0};
 craterPointsBinTempOut = Transpose[
   {tblCratersTemp[[All, 1]], tblCratersTemp[[All, 2]], tblCratersTemp[[All, 3]],
    tblCratersTemp[[All, 4]], tblCratersTemp[[All, 5]], col6}]; (* Return the
  revised table with Kthnn distance values appended as new column *)
 Return[craterPointsBinTempOut];
]; (* End of handle Table Length = 1 error *)
If[thisTableLength == 2,
 col6 = {0, 0};
 craterPointsBinTempOut = Transpose[
   {tblCratersTemp[[All, 1]], tblCratersTemp[[All, 2]], tblCratersTemp[[All, 3]],
    tblCratersTemp[[All, 4]], tblCratersTemp[[All, 5]], col6}]; (* Return the
  revised table with Kthnn distance values appended as new column *)
 Return[craterPointsBinTempOut];
]; (* End of handle Table Length = 2 error *)
For[i = 1, i < (thisTableLength + 1), i++,
 thisPointKthNearestDistanceTemp =
  Nearest[craterPointsBinTemp, craterPointsBinTemp[[i, All]], n];
 (* Get current and nth nearest points *)
 thisPointKthNearestDistance =
  N[EuclideanDistance[thisPointKthNearestDistanceTemp[[1, All]],
    thisPointKthNearestDistanceTemp[[n, All]]]];
 (* Compute the distance between the current and nth nearest points *)
 lstKthnnDistance[i] = thisPointKthNearestDistance;
 (* Append result to holding list *)
]; (* End of for loop *)
(* Programming Note: For loop must also end with ";" within module. *)
col6 = Table[lstKthnnDistance[k], {k, 1, thisTableLength}];
(* Create column of distances *)
```

```
  (* Programming Note: Table command uses length of original table,
  not the length of the vector of computed distances *)
  craterPointsBinTempOut = Transpose[
    {tblCratersTemp[[All, 1]], tblCratersTemp[[All, 2]], tblCratersTemp[[All, 3]],
     tblCratersTemp[[All, 4]], tblCratersTemp[[All, 5]], col6}]
  (* Return the revised table with NthKnn values appended as new column *)
 ]

(* DEBUG NOTE:  Double check the runtime error in
   UpdateTableKthnn function to make sure it is returning the 2nd,
3rd Nearest Neighbor and not the 1st, 2nd Nearest Neighbor. *)
```

```
MakeSummaryBinRow[tblBinNTemp_, tblBinDivTemp_] := Module[
  {lowBinTemp, currentBinTable, lowIntervalTemp,
   upperIntervalTemp, binCountTemp, bin2knnMeanTemp, bin2knnVarTemp,
   bin2knnSDTemp, binSumRowTemp, thistblBinDivision},
  (* tblBinNTemp_ = A table of craters with appended 2n Knn distances
    consisting of six columns: obsIdx, Xcoord, Ycoord, Diameter, BinNumber,
  knn2dDistance; tblBinDivTemp = a list of the bin division intervals;
  Returns a row that summarizes the statistical characteristics of
   the 2nd Knn distance for this bin containing: lowBinInterval,
  upperBinInterval, binCount, binMean, binSD, binVar. *)
  (* Build Bin summary row *)
  thisBinTable = tblBinNTemp;
  thistblBinDivision = tblBinDivTemp;
  lowBinTemp = DeleteDuplicates[Level[thisBinTable[[All, 5]], 1]] ;
  (* Get the bin number from the list. *)
  (* Programming note:  There had better be only one element in this list,
  or abend occurs *) (* This is the lower interval bin number *)
lowIntervalTemp = thistblBinDivision[[lowBinTemp]][[1]];
  (* Get the lower interval bin value *)
(* Program note: Trailing [[1]] extracts list element as a value. *)
  lowBinTemp + 1;
upperIntervalTemp = thistblBinDivision[[lowBinTemp + 1]][[1]];
  (* Get the upper interval bin value *)
binCountTemp = Length[thisBinTable[[All, 6]]]; (* Count of 2dKnn distances *)
bin2knnMeanTemp = Mean[thisBinTable[[All, 6]]] ; (* Mean of 2dKnn distances *)
  (* Handle runtime error for bins that contain only 1 element,
  thus, Var and SD cannot be computed. *)
  If[binCountTemp == 1,
   binSumRowTemp =
    {lowIntervalTemp, upperIntervalTemp, binCountTemp, bin2knnMeanTemp, 0, 0};
   (* Return the revised row with Var = 0 and Sd = 0. *)
   Return[binSumRowTemp ];
  ] ; (* End of handle bin count = 1 error *)
bin2knnVarTemp = Variance[thisBinTable[[All, 6]]];
  (* Variance of 2dKnn distances *)
bin2knnSDTemp = StandardDeviation[thisBinTable[[All, 6]]];
  (* Standard deviation of 2dKnn distances *)
binSumRowTemp = {lowIntervalTemp, upperIntervalTemp,
    binCountTemp, bin2knnMeanTemp, bin2knnVarTemp, bin2knnSDTemp}
 ]
```

```
getPointInBox[tblObsFrameTemp_] := Module[
  {thistblObsFrame, thisPointWorking, LowerLeftX, LowerLeftY,
   LowerRightX, LowerRightY, UpperLeftX, UpperRightX, UpperLeftY, UpperRightY,
   BoxMinLeftX, BoxMinLeftY, BoxMaxRightX, BoxMaxUpperY, bolBotLine,
   bolTopLine, bolLeftLine, bolRightLine, thisPointGood, bolTestPoint},
  (* tblObsFrame = boundaries of four-sided observing frame in dictionary format;
  thisPoint = x-y pair ;
  Returns x-y pair inside rectangular box that encloses four-
   sided observing frame. *)
  thistblObsFrame = tblObsFrameTemp ;
  (* Uses:  bolPointInsideFrame *)
  (* Decompress observing frame dimensions *) LowerLeftX =
   thistblObsFrame[[Flatten[Position[thistblObsFrame, "LowerLeftX"]][[1]], 2]];
LowerLeftY = thistblObsFrame[[
    Flatten[Position[thistblObsFrame, "LowerLeftY"]][[1]], 2]];
LowerRightX = thistblObsFrame[[
    Flatten[Position[thistblObsFrame, "LowerRightX"]][[1]], 2]];
LowerRightY = thistblObsFrame[[
    Flatten[Position[thistblObsFrame, "LowerRightY"]][[1]], 2]];
UpperLeftX = thistblObsFrame[[Flatten[Position[thistblObsFrame, "UpperLeftX"]][[1]],
    2]];
UpperLeftY = thistblObsFrame[[Flatten[Position[thistblObsFrame, "UpperLeftY"]][[1]],
    2]];
UpperRightX = thistblObsFrame[[
    Flatten[Position[thistblObsFrame, "UpperRightX"]][[1]], 2]];
UpperRightY = thistblObsFrame[[
    Flatten[Position[thistblObsFrame, "UpperRightY"]][[1]], 2]];
(* Get the dimensions of the rectangular box that encloses the parallelogram *)
(* Programming note: Compute these once in order to reduce overhead *)
 BoxMinLeftX = Min[UpperLeftX, LowerLeftX];
  BoxMaxRightX = Max[UpperRightX, LowerRightX];
BoxMinLeftY = Min[LowerLeftY, LowerRightY];
  BoxMaxUpperY = Max[UpperLeftY, UpperRightY];
  bolTestPoint  = False;
  While[bolTestPoint  == False, (* Search until a valid point is returned *)
   thisPointWorking = {RandomInteger[{BoxMinLeftX , BoxMaxRightX}],
     RandomInteger[{BoxMinLeftY, BoxMaxUpperY}]};
   bolTestPoint = bolPointInsideFrame[thistblObsFrame, thisPointWorking];
  ];
  Return[thisPointWorking]
 ]

bolPointInsideFrame[tblObsFrameTemp_, thisPointTemp_] := Module[
  {thistblObsFrame, thisPointWorking, LowerLeftX, LowerLeftY,
   LowerRightX, LowerRightY, UpperLeftX, UpperRightX, UpperLeftY,
   UpperRightY, BoxMinLeftX, BoxMinLeftY, BoxMaxRightX, BoxMaxUpperY,
   bolBotLine, bolTopLine, bolLeftLine, bolRightLine, thisPointGood},
```

```
  (* tblObsFrame = boundaries of four-sided observing frame in dictionary format;
  thisPoint = x-y pair ;
  Returns true if point is within observing frame,
  and False if it is not within observing frame. *)
  (* Used By:  getPointInBox *)
  thistblObsFrame = tblObsFrameTemp ;
  thisPointWorking = thisPointTemp;
  (* Decompress observing frame dimensions *)LowerLeftX =
   thistblObsFrame[[Flatten[Position[thistblObsFrame, "LowerLeftX"]][[1]], 2]];
LowerLeftY = thistblObsFrame[[
    Flatten[Position[thistblObsFrame, "LowerLeftY"]][[1]], 2]];
LowerRightX = thistblObsFrame[[
    Flatten[Position[thistblObsFrame, "LowerRightX"]][[1]], 2]];
LowerRightY = thistblObsFrame[[
    Flatten[Position[thistblObsFrame, "LowerRightY"]][[1]], 2]];
UpperLeftX = thistblObsFrame[[Flatten[Position[thistblObsFrame, "UpperLeftX"]][[1]],
    2]];
UpperLeftY = thistblObsFrame[[Flatten[Position[thistblObsFrame, "UpperLeftY"]][[1]],
    2]];
UpperRightX = thistblObsFrame[[
    Flatten[Position[thistblObsFrame, "UpperRightX"]][[1]], 2]];
UpperRightY = thistblObsFrame[[
    Flatten[Position[thistblObsFrame, "UpperRightY"]][[1]], 2]];
(* Get the dimensions of the rectangular box that encloses the parallelogram *)
(* Programming note: Compute these once in order to reduce overhead *)
 BoxMinLeftX = Min[UpperLeftX, LowerLeftX];
  BoxMaxRightX = Max[UpperRightX, LowerRightX];
BoxMinLeftY = Min[LowerLeftY, LowerRightY];
  BoxMaxUpperY = Max[UpperLeftY, UpperRightY];
  (* Test if the point is inside the four-sided observing frame. *)
  (* Two point line forumula for top-bottom line tests that test y:  y - y1 =
   ((y2-y1)/(x2-x1))*(x-x1) →   y < ((y2-y1)/(x2-x1))*(x-x1) + y1 *)
  (*  *)
  (* Step A: Test top line Y line using the two point line formula *)
  If[ (* Condition *)
   thisPointWorking[[2]] < (((UpperRightY - UpperLeftY) / (UpperRightX - UpperLeftX)) *
       (thisPointWorking[[1]] * UpperLeftX)) + UpperLeftY
   , (* Case: True *) bolTopLine = True , (* Case: False *)
   bolTopLine = False ];
  (*  *)
  (* Step B: Test bottom line Y line using the two point line formula *)
  If[ (* Condition *)
   (* Test bottom line Y line using the two point line formula *)
   thisPointWorking[[2]] > ( ((LowerRightY - LowerLeftY) / (LowerRightX - LowerLeftX)) *
       (thisPointWorking[[1]] * LowerLeftX)) + LowerLeftY
   , (* Case: True *) bolBotLine = True , (* Case: False *)
```

```
 bolBotLine = False ];
(* Two point line forumula for left-right line tests that test x:
    y - y1 = ((y2-y1)/(x2-x1))*(x-x1) → x =
  ((x2-x1)/(y2-y1))*(y-y1)+x1 →  x < ((x2-x1)/(y2-y1))*(y-y1)+x1 *)
(*  *)
(* Step C: Test left x-y line using the two point line formula *)
If[ (* Condition *)
 (* Error handle = rectangular box *) UpperLeftY == LowerLeftY ,
  (* Case: True - Nested If - Simple formula for x>x1 *)
 If[ thisPointWorking[[1]] > UpperLeftX  ,
  bolLeftLine = True  , bolLeftLine = False ]
 , (* Case: False - Nested If - Left line is oblique, use two line formula *)
 If[
   thisPointWorking[[1]] > (((UpperLeftX - LowerLeftX) / (UpperLeftY - LowerLeftY)) *
       (thisPointWorking[[2]] - LowerLeftY)) + LowerLeftX
   , (* Case: True *) bolLeftLine = True
   , (* Case: False *) bolLeftLine = False ];
];
(*  *)
(* Step D: Test right x-y line using the two point line formula *)
If[ (* Condition *)
 (* Error handle = rectangular box *) UpperRightX == LowerRightX ,
  (* Case: True - Nested If - Simple forumula for x>x1 *)
 If[ thisPointWorking[[1]] < UpperRightX  ,
  bolRightLine = True  , bolRightLine = False ]
 , (* Case: False - Nested If - Left line is oblique, use two line formula *)
 If[thisPointWorking[[1]] <
   (((UpperRightX - LowerRightX) / (UpperRightY - LowerRightY)) *
      (thisPointWorking[[2]] - LowerRightY)) + LowerRightX
  , (* Case: True *) bolRightLine = True
  , (* Case: False *) bolRightLine = False ]
];
(*  *)
(* Step E: If this point inside the observation frame? *)
thisPointGood = If[ (* Condition *)
  (bolTopLine && bolBotLine && bolLeftLine && bolRightLine ) == True
  , (* Case: True *) True , (* Case: False *)  False ]
]
```

```
getSimulatedBinKthnn[tblObsFrameTemp_,
  binSizeTemp_, binNumberTemp_, globalKnnTimesTemp_] := Module[
  {thistblObsFrame, thisbinSize, thisbinNumber, pointsSimulatedBin,
   tblSimulatedBin, tblKthnnSimBin, meanSimBin, varSimBin},
  (* tblObsFrame = boundaries of four-sided observing frame in dictionary format;
  binSizeTemp = the size of the simulated bin ;
  globalKnnTimesTemp is the number of the nearest neighbor sought,
  and it is set from a global variable.
     Returns a table of simulated bin of points inside a four-
   sided observing frame in six column format -
       obsId, coordX, coordY, fake diameter, bin no, kth knn distance. *)
  thistblObsFrame = tblObsFrameTemp ;
  thisbinSize = binSizeTemp;
  thisbinNumber = binNumberTemp;
  (* Uses:  getPointInBox, UpdateTableKthnn *)
  (* Make simulation of bin *)
  pointsSimulatedBin = Table[getPointInBox[thistblObsFrame], {i, 1, thisbinSize}];
  (* Programming note: Consider error check in simulation function for a
     point that might already be in the box should never occur twice. *)
  colA1 = Table[i, {i, 1, thisbinSize}]; (* obs id for a simulated bin table *)
colA2 = pointsSimulatedBin[[All, 1]]; (* coordX for a simulated bin table *)
colA3 = pointsSimulatedBin[[All, 2]]; (* coordY id for a simulated bin table *)
colA4 = Table[0, {i, 1, thisbinSize}];
  (* fake diameter for a simulated bin table *)
colA5 = Table[thisbinNumber, {i, 1, thisbinSize}];
  (* bin no for a simulated bin table *)
tblSimulatedBin = Transpose[{colA1, colA2, colA3, colA4, colA5}];
  (* assemble simulated bin *)
tblKthnnSimBin = UpdateTableKthnn[tblSimulatedBin, globalKnnTimes]
]
```

```
getSimulatedBinKthnnMean[tblObsFrameTemp_,
  binSizeTemp_, binNumberTemp_, globalKnnTimesTemp_] := Module[
  {tblKthnnSimBin, meanSimBin, varSimBin},
  (* tblObsFrame = boundaries of four-sided observing frame in dictionary format;
  binSizeTemp = the size of the simulated bin ;
  globalKnnTimesTemp is the number of the nearest neighbor sought,
  and it is set from a global variable.
     Returns a table of the mean and variance of a simulated bin of points
    inside a four-sided observing frame in two column format - {mean,var}. *)
  (* Uses:  getPointInBox, UpdateTableKthnn, getSimulatedBinKthnn *)
  (* Make simulation of bin *)
  (* Wrap getSimulatedBinKthnn *)
  tblKthnnSimBin =  getSimulatedBinKthnn[
    tblObsFrameTemp, binSizeTemp, binNumberTemp, globalKnnTimesTemp];
  Return[{Mean[tblKthnnSimBin[[All, 6]]], Variance[tblKthnnSimBin[[All, 6]]]}]
]
```

Get data

```
tblCratersRaw = Import["Craters.csv", "Data"] ; (* Get raw data table *)
```

```
tblCratersRaw = Drop[tblCratersRaw, 1]; (* Delete the header row *)
```

```
(* Cleanup row based on unique table characteristics *)
```

```
tblCratersRaw = Drop[tblCratersRaw, -5]; (* Delete the last 5 blank rows *)
```

```
Length[tblCratersRaw]  (* Check raw data table length *)
```

```
890
```

```
(* Table display disabled for production code. *)
```

```
Grid[tblCratersRaw] ;(* Grid and display raw data table *)
```

Build working table of crater positions

```
lastRow = Length[tblCratersRaw];
(* Clip blank trailing rows from raw data table. *)
```

```
col1 = Table[tblCratersRaw[[i, 1]], {i, 1, lastRow}]; (* Get index number *)
```

```
col2 = Table[tblCratersRaw[[i, 2]], {i, 1, lastRow}]; (* Get x coordinate *)
```

```
col3 = Table[tblCratersRaw[[i, 3]], {i, 1, lastRow}]; (* Get y coordinate *)
```

```
col4 = Table[tblCratersRaw[[i, 15]], {i, 1, lastRow}] ;
(* Get crater diameter MinFeret's diameter *)
```

Compute the 2 n bin number for all the crater diameters using function

Get2NBinNumber, adjusted for minimum crater diameter.

```
minCraterDiaHold = Min[col4]
 (* Compute the minimum crater diameter in the dataset. *)
```

```
4
```

```
diameterDataTemp = Transpose[
   {Table[col4[[i]], {i, 1, lastRow}], Table[minCraterDiaHold, {i, 1, lastRow}]}];
```

```
col5 = Map[Get2NBinNumber, diameterDataTemp];
```

```
Length[col5]
```

```
890
```

Create and display the cleaned, working data table.

```
tblCraters = Transpose[{col1, col2, col3, col4, col5}];
```

```
gridCraters = Grid[tblCraters];
```

```
(* Table display disabled for production code. *)
```

```
gridCraters;
```

```
craterPointsAll = Transpose[{tblCraters[[All, 2]], tblCraters[[All, 3]]}];
```

Build list of bin intervals.

```
tblBinDivisions = Table[2^i + minCraterDiaHold, {i, 1, Max[tblCraters[[All, 5]]]}];
(* Make a list of the bin divisions by crater diameter *)
```

```
tblBinDivisions = Insert[tblBinDivisions, minCraterDiaHold, 1];
```

Clean up

```
 col1 = Null; col2 = Null; col3 = Null; col4 = Null;
col5 = Null; diameterDataTemp = Null; lastRow = Null;
(* Programming note - removal of values can affect debugging above this
  point. Comment out nulling statements before debugging above this point. *)
```

Persisted data objects at this point:

tblCraters = cleaned, selected columns of crater data in a table object.
    Contains: Index, Xcoord, Ycoord, Diameter(Minimum Feret), 2n Bin number.
Grid[tblCratersRaw] = Raw imported data table in grid format.
    Contains: Field list is above.
minCraterDiaHold = Smallest diameter crater in the crater data table (tblCraters).
tblBinDivisions = List of bin divisions.

Build plot panel for Kolmogorov-Smirnov Test.

Plot bin histogram.

```
bincountsTbl = BinCounts[tblCraters[[All, 5]]] ;
(* Plot number of points in each 2n bin - 1 through 6 *)

bincountsTbl = Drop[bincountsTbl, {1}]
```

```
{347, 283, 187, 58, 14, 1}
```

```
(* Plot display disabled for production code. *)

a = Histogram[tblCraters[[All, 5]], ChartStyle → {White},
   Frame → True, FrameLabel → {"A", " "}]; (* Plot the points *)
```

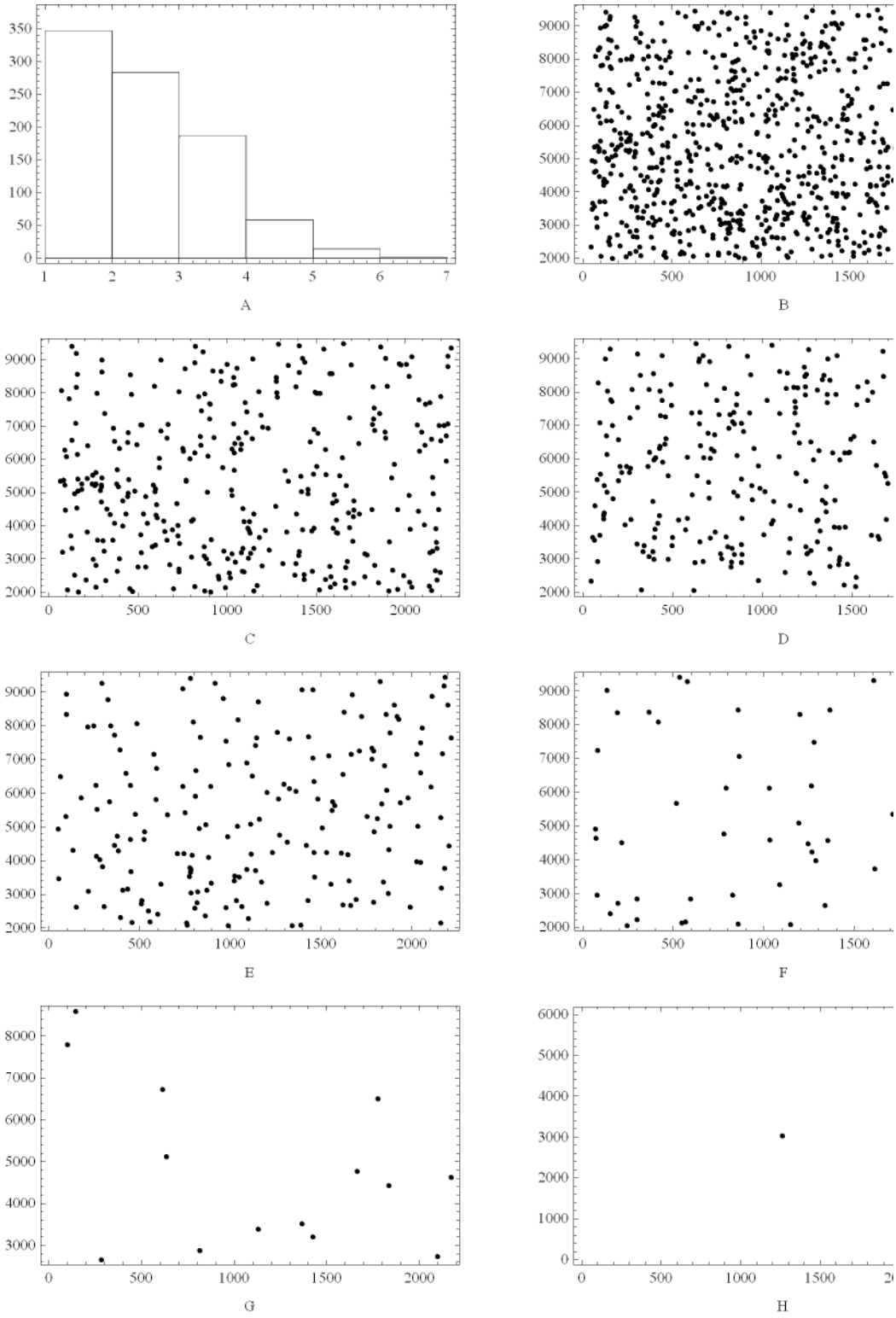
Plot distribution of crater points for all craters and for individual bins.

```
positionsBinN = Flatten[Position[tblCraters[[All, 5]], 1]];

tblCratersBin1 = tblCraters[[positionsBinN, All]];

craterPointsBin1 = Transpose[{tblCratersBin1[[All, 2]], tblCratersBin1[[All, 3]]}];

positionsBinN = Flatten[Position[tblCraters[[All, 5]], 2]];

tblCratersBin2 = tblCraters[[positionsBinN, All]];

craterPointsBin2 = Transpose[{tblCratersBin2[[All, 2]], tblCratersBin2[[All, 3]]}];

positionsBinN = Flatten[Position[tblCraters[[All, 5]], 3]];

tblCratersBin3 = tblCraters[[positionsBinN, All]];

craterPointsBin3 = Transpose[{tblCratersBin3[[All, 2]], tblCratersBin3[[All, 3]]}];

positionsBinN = Flatten[Position[tblCraters[[All, 5]], 4]];

tblCratersBin4 = tblCraters[[positionsBinN, All]];

craterPointsBin4 = Transpose[{tblCratersBin4[[All, 2]], tblCratersBin4[[All, 3]]}];

positionsBinN = Flatten[Position[tblCraters[[All, 5]], 5]];

tblCratersBin5 = tblCraters[[positionsBinN, All]];

craterPointsBin5 = Transpose[{tblCratersBin5[[All, 2]], tblCratersBin5[[All, 3]]}];

positionsBinN = Flatten[Position[tblCraters[[All, 5]], 6]];

tblCratersBin6 = tblCraters[[positionsBinN, All]];

craterPointsBin6 = Transpose[{tblCratersBin6[[All, 2]], tblCratersBin6[[All, 3]]}];

(* Plot display disabled for production code. *)
```

```
b = ListPlot[craterPointsAll,
   PlotStyle → Directive[Black] , PlotStyle → {PointSize[0.01]},
   Frame → True , FrameLabel → {"B", " "}] (* Plot the points *);

(* Plot display disabled for production code. *)

c = ListPlot[craterPointsBin1,
   PlotStyle → Directive[Black] , PlotStyle → {PointSize[0.01]},
   Frame → True , FrameLabel → {"C", " "}] ;(* Plot the points *)

(* Plot display disabled for production code. *)

d = ListPlot[craterPointsBin2,
   PlotStyle → Directive[Black] , PlotStyle → {PointSize[0.01]},
   Frame → True , FrameLabel → {"D", " "}] ;(* Plot the points *)

(* Plot display disabled for production code. *)

e = ListPlot[craterPointsBin3,
   PlotStyle → Directive[Black] , PlotStyle → {PointSize[0.01]},
   Frame → True , FrameLabel → {"E", " "}] ;(* Plot the points *)

(* Plot display disabled for production code. *)

f = ListPlot[craterPointsBin4,
   PlotStyle → Directive[Black] , PlotStyle → {PointSize[0.01]},
   Frame → True , FrameLabel → {"F", " "}] ;(* Plot the points *)

(* Plot display disabled for production code. *)

g = ListPlot[craterPointsBin5,
   PlotStyle → Directive[Black] , PlotStyle → {PointSize[0.01]},
   Frame → True , FrameLabel → {"G", " "}] ;(* Plot the points *)

(* Plot display disabled for production code. *)

h = ListPlot[craterPointsBin6 ,
   PlotStyle → Directive[Black], PlotStyle → {PointSize[0.01]},
   Frame → True , FrameLabel → {"H", " "}] ;(* Plot the points *)
```

Assemble and export the plot panel of actual crater measurements.

```
binImageCombinedActualJpg =
 GraphicsGrid[{{a, b}, {c, d}, {e, f}, {g, h}}, ImageSize → {800, 800}]
```




```
Export["FigBinsActual.jpg", binImageCombinedActualJpg];

(* Programming note:  SVG formats are not rendering properly. *)

binImageCombinedActualSvg = GraphicsGrid[{{a, b}, {c, d}, {e, f}, {g, h}}];
```

```
Export["FigBinsActual.svg", binImageCombinedActualSvg];
```

Build and 2nd Knn distance tables.

Update crater bin tables with 2nd Knn distances.

```
tblKthnnBin1 = UpdateTableKthnn[tblCratersBin1, globalKnnTimes];

tblKthnnBin2 = UpdateTableKthnn[tblCratersBin2, globalKnnTimes];

tblKthnnBin3 = UpdateTableKthnn[tblCratersBin3, globalKnnTimes];

tblKthnnBin4 = UpdateTableKthnn[tblCratersBin4, globalKnnTimes];

tblKthnnBin5 = UpdateTableKthnn[tblCratersBin5, globalKnnTimes];

tblKthnnBin6 = UpdateTableKthnn[tblCratersBin6, globalKnnTimes];
(* Includes testing 1 point error handling *)
```

Test normality of bins for actual data.

```
(* Normality of Bin 1 *)

plothisttblKthnnBin1 = Histogram[tblKthnnBin1[[All, 6]],
   ChartStyle → LightGray, Frame → True, FrameLabel → {"A", " "}] ;

plotqqtblKthnnBin1 =
  Labeled[QuantilePlot[tblKthnnBin1[[All, 6]], PlotStyle → Directive[Black ] ] ,
   Shapiro Wilk == ShapiroWilkTest[tblKthnnBin1[[All, 6]]],
   LabelStyle → Directive[FontFamily → "Times"]];

ShapiroWilkTest[tblKthnnBin1[[All, 6]]] (* Shaprio Wilk for Actual Data Bin 1 *)
```

4.00821×10^{-7}

```
(* Normality of Bin 2 *)

plothisttblKthnnBin2 = Histogram[tblKthnnBin2[[All, 6]],
   ChartStyle → LightGray , Frame → True, FrameLabel → {"B", " "}] ;

plotqqtblKthnnBin2 =
  Labeled[ QuantilePlot[tblKthnnBin2[[All, 6]], PlotStyle → Directive[Black ] ],
   Shapiro Wilk == ShapiroWilkTest[tblKthnnBin2[[All, 6]]],
   LabelStyle → Directive[FontFamily → "Times"]];

ShapiroWilkTest[tblKthnnBin2[[All, 6]]] (* Shaprio Wilk for Actual Data Bin 2 *)
```

6.36349×10^{-9}

```
(* Normality of Bin 3 *)

plothisttblKthnnBin3 = Histogram[tblKthnnBin3[[All, 6]],
   ChartStyle → LightGray , Frame → True, FrameLabel → {"C", " "}] ;
```

```
plotqqtblKthnnBin3 =
  Labeled[QuantilePlot[tblKthnnBin3[[All, 6]], PlotStyle → Directive[Black ] ],
   Shapiro Wilk == ShapiroWilkTest[tblKthnnBin3[[All, 6]]],
   LabelStyle → Directive[FontFamily → "Times"]];

ShapiroWilkTest[tblKthnnBin3[[All, 6]]] (* Shaprio Wilk for Actual Data Bin 3 *)
```

```
0.0000136925
```

```
(* Normality of Bin 4 *)

plothisttblKthnnBin4 = Histogram[tblKthnnBin4[[All, 6]],
   ChartStyle → LightGray , Frame → True, FrameLabel → {"D", " "}] ;

plotqqtblKthnnBin4 =
  Labeled[QuantilePlot[tblKthnnBin4[[All, 6]], PlotStyle → Directive[Black ] ],
   Shapiro Wilk == ShapiroWilkTest[tblKthnnBin4[[All, 6]]],
   LabelStyle → Directive[FontFamily → "Times"]];

ShapiroWilkTest[tblKthnnBin4[[All, 6]]] (* Shaprio Wilk for Actual Data Bin 4 *)
```

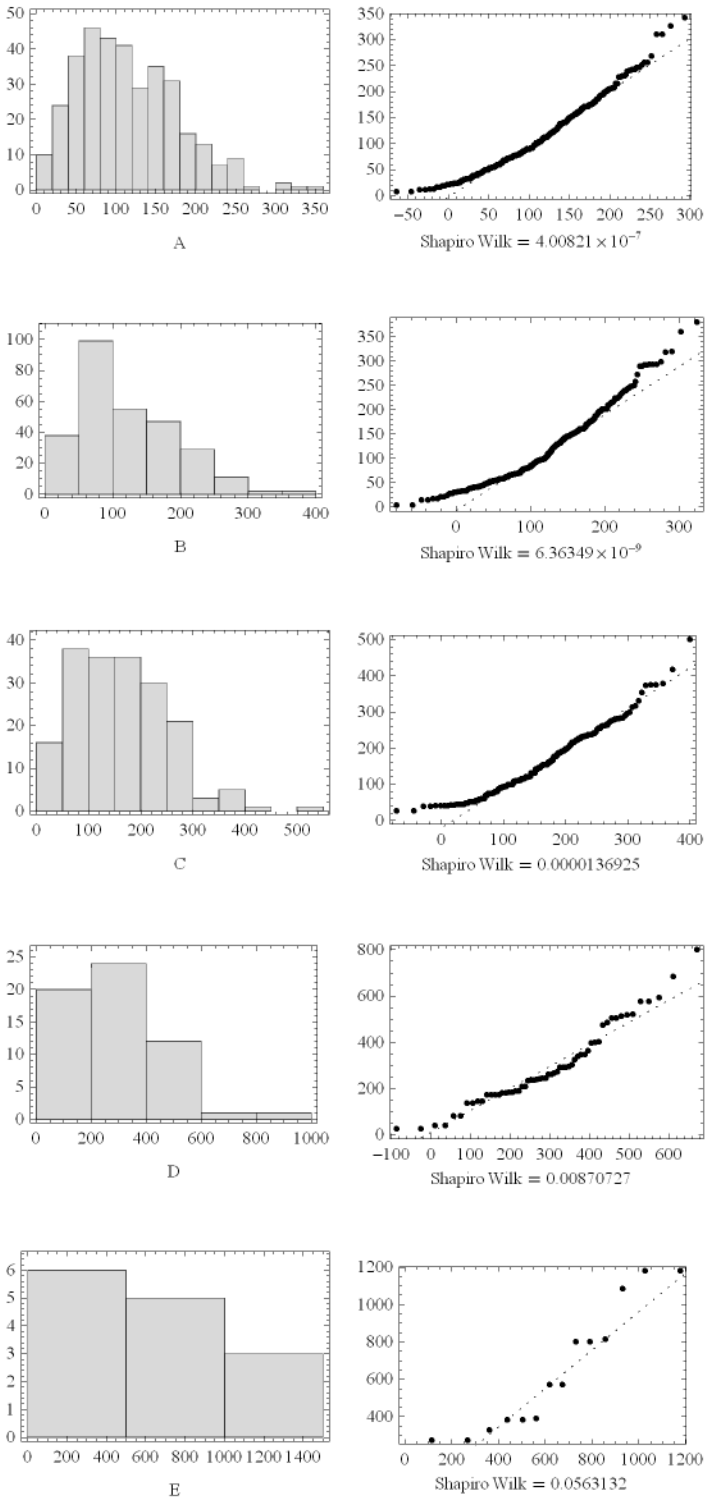
```
0.00870727
```

```
(* Normality of Bin 5 *)

plothisttblKthnnBin5 = Histogram[tblKthnnBin5[[All, 6]],
   ChartStyle -> LightGray, Frame → True, FrameLabel → {"E", " "}] ;

plotqqtblKthnnBin5 =
  Labeled[QuantilePlot[tblKthnnBin5[[All, 6]], PlotStyle → Directive[Black ] ],
   Shapiro Wilk == ShapiroWilkTest[tblKthnnBin5[[All, 6]]] ,
   LabelStyle → Directive[FontFamily → "Times"]];

ShapiroWilkTest[tblKthnnBin5[[All, 6]]] (* Shaprio Wilk for Actual Data Bin 5 *)
```

```
0.0563132
```

- Assemble and export the plot panel concerning normality of actual data.

```
binImageCombinedActualNormalityJpg =
 GraphicsGrid[{{plothisttblKthnnBin1, plotqqtblKthnnBin1},
   {plothisttblKthnnBin2, plotqqtblKthnnBin2}, {plothisttblKthnnBin3,
    plotqqtblKthnnBin3}, {plothisttblKthnnBin4, plotqqtblKthnnBin4},
   {plothisttblKthnnBin5, plotqqtblKthnnBin5}}, ImageSize → {800, 800}]
```




```
Export["FigNormalityActualBins.jpg", binImageCombinedActualNormalityJpg];
```

```
(* Programming
 note:  SVG format requires larger y grid dimension in order to render properly,
but svgs are not rendering properly. *)

binImageCombinedActualNormalitySvg =
  GraphicsGrid[{{plothisttblKthnnBin1, plotqqtblKthnnBin1},
    {plothisttblKthnnBin2, plotqqtblKthnnBin2}, {plothisttblKthnnBin3,
     plotqqtblKthnnBin3}, {plothisttblKthnnBin4, plotqqtblKthnnBin4},
    {plothisttblKthnnBin5, plotqqtblKthnnBin5}}, ImageSize → {800, 1200}];

Export["FigNormalityActualBins.svg", binImageCombinedActualNormalitySvg];
```

Export crater bin tables to tab delimited files and bin division list for statistical evaluation in R.

```
tblHeader = {"obsId", "x", "y", "dia", "bin", "kthnnDistance"};
(* Make the header table *)

 (* Export bin 1 data *)

tblKthnnBin1Temp = tblKthnnBin1;

tblKthnnBin1Temp = Insert[ tblKthnnBin1Temp, tblHeader, 1];

Export["tblKthnnBin1.txt", tblKthnnBin1Temp, "Table"];

 (* Export bin 2 data *)

tblKthnnBin2Temp = tblKthnnBin2;

tblKthnnBin2Temp = Insert[ tblKthnnBin2Temp, tblHeader, 1];

Export["tblKthnnBin2.txt", tblKthnnBin2Temp, "Table"];

(* Export bin 3 data *)

tblKthnnBin3Temp = tblKthnnBin3;

tblKthnnBin3Temp = Insert[ tblKthnnBin3Temp, tblHeader, 1];

Export["tblKthnnBin3.txt", tblKthnnBin3Temp, "Table"];

(* Export bin 4 data *)

tblKthnnBin4Temp = tblKthnnBin4;

tblKthnnBin4Temp = Insert[ tblKthnnBin4Temp, tblHeader, 1];

Export["tblKthnnBin4.txt", tblKthnnBin4Temp, "Table"];

(* Export bin 5 data *)

tblKthnnBin5Temp = tblKthnnBin5;

tblKthnnBin5Temp = Insert[ tblKthnnBin5Temp, tblHeader, 1];

Export["tblKthnnBin5.txt", tblKthnnBin5Temp, "Table"];
```

```
(* Export bin 6 data *)

tblKthnnBin6Temp = tblKthnnBin6;

tblKthnnBin6Temp = Insert[ tblKthnnBin6Temp, tblHeader, 1];

Export["tblKthnnBin6.txt", tblKthnnBin6Temp, "Table"];

(* Export bin division list *)

Export["knnBinDivisions.txt", tblBinDivisions, "Table"];
```

Build and export the kth nearest neighbor summary table by bin.

Make summary table rows

```
tblHeaderBinSum = {"LowBinInterval", "UpperBinInterval", "binCount",
   "binMeanKthnn", "binVarKthnn", "binSdKthnn"}; (* Make the header table *)

bin1SumRowTemp = MakeSummaryBinRow[tblKthnnBin1, tblBinDivisions];
(* Build Bin 1 summary row *)

bin2SumRowTemp = MakeSummaryBinRow[tblKthnnBin2, tblBinDivisions];
(* Build Bin 2 summary row *)

bin3SumRowTemp = MakeSummaryBinRow[tblKthnnBin3, tblBinDivisions];
(* Build Bin 3 summary row *)

bin4SumRowTemp = MakeSummaryBinRow[tblKthnnBin4, tblBinDivisions];
(* Build Bin 4 summary row *)

bin5SumRowTemp = MakeSummaryBinRow[tblKthnnBin5, tblBinDivisions];
(* Build Bin 5 summary row *)

bin6SumRowTemp = MakeSummaryBinRow[tblKthnnBin6, tblBinDivisions];
(* Build Bin 6 summary row *)
```

Assemble rows into a table and export.

```
tblBinSummary = {tblHeaderBinSum, bin1SumRowTemp, bin2SumRowTemp,
   bin3SumRowTemp, bin4SumRowTemp, bin5SumRowTemp, bin6SumRowTemp};

Grid[tblBinSummary]
```

LowBinInterval	UpperBinInterval	binCount	binMeanKthnn	binVarKthnn	binSdKthnn
4	6	347	114.78	3879.02	62.2818
6	8	283	122.352	5171.6	71.9138
8	12	187	163.443	7817.63	88.4174
12	20	58	293.109	28 720.2	169.47
20	36	14	645.999	110 674.	332.677
36	68	1	0	0	0

```
Export["knnBinSummary.txt", tblBinSummary, "Table"];

Export["knnBinSummary.tex", Grid[tblBinSummary], "LaTex"];
```

Cleanup

```
tblKthnnBin1Temp = Null; tblKthnnBin2Temp = Null; tblKthnnBin3Temp = Null;
tblKthnnBin4Temp = Null; tblKthnnBin5Temp = Null; tblKthnnBin6Temp = Null;
bin1SumRowTemp = Null; bin2SumRowTemp = Null; bin3SumRowTemp = Null;
bin4SumRowTemp = Null; bin5SumRowTemp = Null; bin6SumRowTemp = Null;
```

Persist object tblBinSummary.

Make simulated points and simulated crater table.

Get dimensions of four-sided observation frame from file.

```
tblObsFrame = Import["CraterBoxDim.txt", "Data"] ; (* Get raw data table *)
```

```
tblObsFrame
```

```
{{LowerLeftX, 25}, {LowerLeftY, 2000}, {LowerRightX, 2217}, {LowerRightY, 2000},
 {UpperLeftX, 81}, {UpperLeftY, 9500}, {UpperRightX, 2270}, {UpperRightY, 9500}}
```

Generate simulated bins of points.

```
tblBinSummary
```

```
{{LowBinInterval, UpperBinInterval, binCount, binMeanKthnn, binVarKthnn, binSdKthnn},
 {4, 6, 347, 114.78, 3879.02, 62.2818}, {6, 8, 283, 122.352, 5171.6, 71.9138},
 {8, 12, 187, 163.443, 7817.63, 88.4174}, {12, 20, 58, 293.109, 28720.2, 169.47},
 {20, 36, 14, 645.999, 110674., 332.677}, {36, 68, 1, 0, 0, 0}}
```

Make a single frame of simulated points in each bin and export to a figure.

- Build tables of the simulated points in bins.

```
tblSimulatedKthnnBin1 = getSimulatedBinKthnn[tblObsFrame,
   tblBinSummary[[2, 3]], 1, globalKnnTimes] ;(* Simulate bin 1 *)
```

```
craterPointsSimulatedBin1 =
  Transpose[{tblSimulatedKthnnBin1[[All, 2]], tblSimulatedKthnnBin1[[All, 3]]}];
```

```
tblSimulatedKthnnBin2 = getSimulatedBinKthnn[tblObsFrame,
   tblBinSummary[[3, 3]], 2, globalKnnTimes]; (* Simulate bin 2 *)
```

```
craterPointsSimulatedBin2 =
  Transpose[{tblSimulatedKthnnBin2[[All, 2]], tblSimulatedKthnnBin2[[All, 3]]}];
```

```
tblSimulatedKthnnBin3 = getSimulatedBinKthnn[tblObsFrame,
   tblBinSummary[[4, 3]], 3, globalKnnTimes]; (* Simulate bin 3 *)
```

```
craterPointsSimulatedBin3 =
  Transpose[{tblSimulatedKthnnBin3[[All, 2]], tblSimulatedKthnnBin3[[All, 3]]}];
```

```
tblSimulatedKthnnBin4 = getSimulatedBinKthnn[tblObsFrame,
   tblBinSummary[[5, 3]], 4, globalKnnTimes];(* Simulate bin 4 *)
```

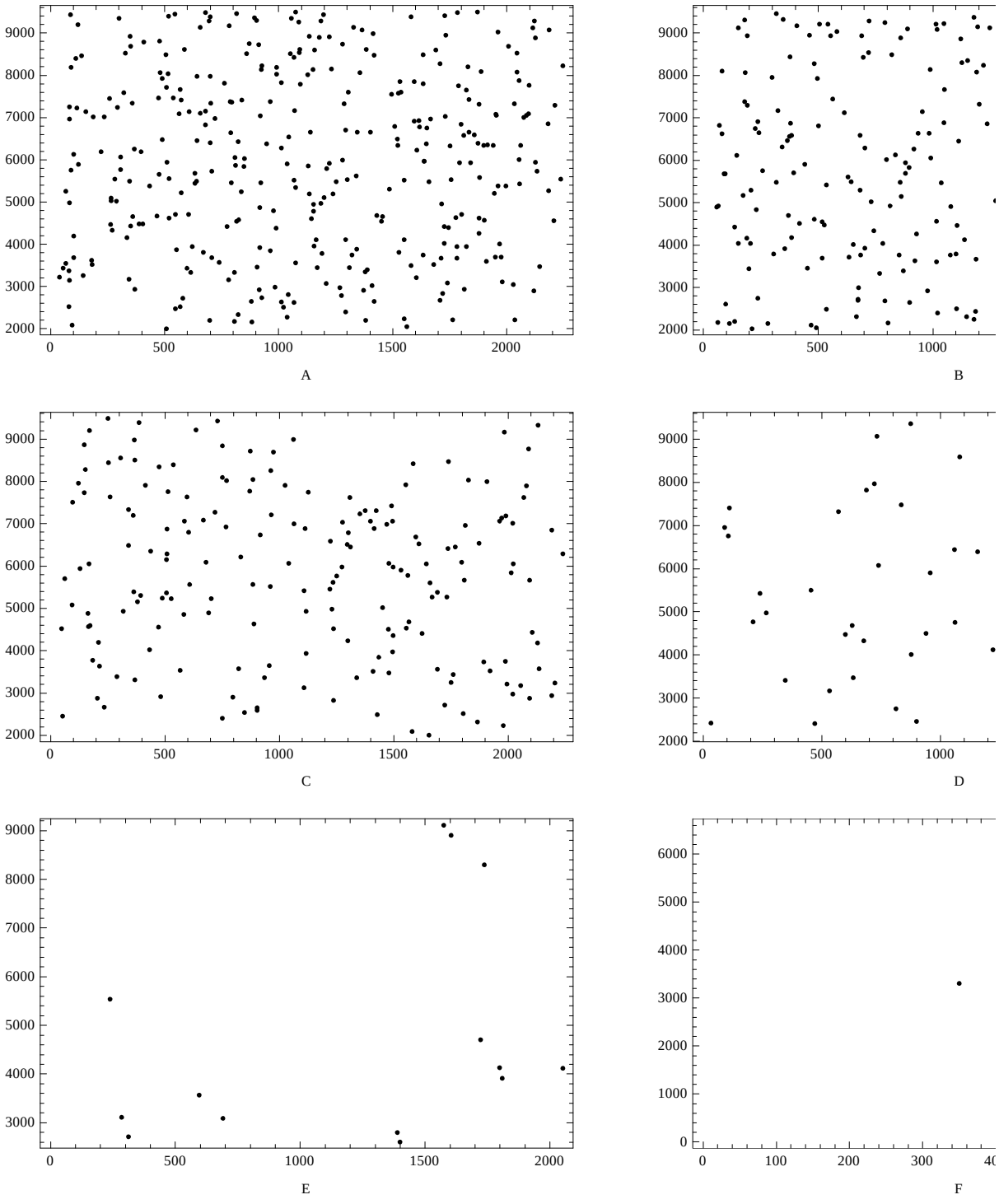
```
craterPointsSimulatedBin4 =
  Transpose[{tblSimulatedKthnnBin4[[All, 2]], tblSimulatedKthnnBin4[[All, 3]]}];

tblSimulatedKthnnBin5 = getSimulatedBinKthnn[tblObsFrame,
   tblBinSummary[[6, 3]], 5, globalKnnTimes]; (* Simulate bin 5 *)

craterPointsSimulatedBin5 =
  Transpose[{tblSimulatedKthnnBin5[[All, 2]], tblSimulatedKthnnBin5[[All, 3]]}];

tblSimulatedKthnnBin6 = getSimulatedBinKthnn[tblObsFrame,
   tblBinSummary[[7, 3]], 6, globalKnnTimes]; (* Simulate bin 6 *)

craterPointsSimulatedBin6 =
  Transpose[{tblSimulatedKthnnBin6[[All, 2]], tblSimulatedKthnnBin6[[All, 3]]}];
```

- Plot the simulated bin points.

```
a = ListPlot[craterPointsSimulatedBin1,
   PlotStyle → Directive[Black] , PlotStyle → {PointSize[0.01]},
   Frame → True , FrameLabel → {"A", " "}]; (* Plot the points *)

b = ListPlot[craterPointsSimulatedBin2,
   PlotStyle → Directive[Black] , PlotStyle → {PointSize[0.01]},
   Frame → True , FrameLabel → {"B", " "}] ;(* Plot the points *)

c = ListPlot[craterPointsSimulatedBin3,
   PlotStyle → Directive[Black] , PlotStyle → {PointSize[0.01]},
   Frame → True , FrameLabel → {"C", " "}];(* Plot the points *)

d = ListPlot[craterPointsSimulatedBin4,
   PlotStyle → Directive[Black] , PlotStyle → {PointSize[0.01]},
   Frame → True , FrameLabel → {"D", " "}];(* Plot the points *)

e = ListPlot[craterPointsSimulatedBin5,
   PlotStyle → Directive[Black] , PlotStyle → {PointSize[0.01]},
   Frame → True , FrameLabel → {"E", " "}];(* Plot the points *)

f = ListPlot[craterPointsSimulatedBin6,
   PlotStyle → Directive[Black] , PlotStyle → {PointSize[0.01]},
   Frame → True , FrameLabel → {"F", " "}];(* Plot the points *)
```

- Assemble and export the plot panel of simulated points.

```
binImageCombinedJpg = GraphicsGrid[{{a, b}, {c, d}, {e, f}}, ImageSize → {800, 800}]
```




```
Export["FigSimulateOneBin.jpg", binImageCombinedJpg];
```

```
(* Programming
 note:  SVG format requires larger y grid dimension in order to render properly,
but svgs are not rendering properly. *)

binImageCombinedSvg = GraphicsGrid[{{a, b}, {c, d}, {e, f}, {g, h}}];

Export["FigSimulateOneBin.svg", binImageCombinedSvg];
```

Test normality of bins for simulated data.

```
(* Normality of Bin 1 *)

plotSimhisttblKthnnBin1 = Histogram[tblSimulatedKthnnBin1[[All, 6]],
   ChartStyle → White , Frame → True, FrameLabel → {"A", " "}] ;

plotSimqqtblKthnnBin1 = Labeled[
   QuantilePlot[tblSimulatedKthnnBin1[[All, 6]], PlotStyle → Directive[Black ] ] ,
   Shapiro Wilk == ShapiroWilkTest[tblSimulatedKthnnBin1[[All, 6]]],
   LabelStyle → Directive[FontFamily → "Times"]];

ShapiroWilkTest[tblSimulatedKthnnBin1[[All, 6]]]
(* Shaprio Wilk for Actual Data Bin 1 *)
```

2.1993×10^{-11}

```
(* Normality of Bin 2 *)

plotSimhisttblKthnnBin2 = Histogram[tblSimulatedKthnnBin2[[All, 6]],
   ChartStyle → White, Frame → True, FrameLabel → {"B", " "}] ;

plotSimqqtblKthnnBin2 = Labeled[
   QuantilePlot[tblSimulatedKthnnBin2[[All, 6]], PlotStyle → Directive[Black ] ],
   Shapiro Wilk == ShapiroWilkTest[tblSimulatedKthnnBin2[[All, 6]]] ,
   LabelStyle → Directive[FontFamily → "Times"]];

ShapiroWilkTest[tblSimulatedKthnnBin2[[All, 6]]]
(* Shaprio Wilk for Actual Data Bin 2 *)
```

1.21631×10^{-6}

```
(* Normality of Bin 3 *)

plotSimhisttblKthnnBin3 = Histogram[tblSimulatedKthnnBin3[[All, 6]],
   ChartStyle → White , Frame → True, FrameLabel → {"C", " "}] ;

plotSimqqtblKthnnBin3 = Labeled[
   QuantilePlot[tblSimulatedKthnnBin3[[All, 6]], PlotStyle → Directive[Black ] ],
   Shapiro Wilk == ShapiroWilkTest[tblSimulatedKthnnBin3[[All, 6]]],
   LabelStyle → Directive[FontFamily → "Times"]];

ShapiroWilkTest[tblSimulatedKthnnBin3[[All, 6]]]
(* Shaprio Wilk for Actual Data Bin 3 *)
```

0.0000682196

```
(* Normality of Bin 4 *)

plotSimhisttblKthnnBin4 = Histogram[tblSimulatedKthnnBin4[[All, 6]],
   ChartStyle → White , Frame → True, FrameLabel → {"D", " "}] ;

plotSimqqtblKthnnBin4 = Labeled[
   QuantilePlot[tblSimulatedKthnnBin4[[All, 6]], PlotStyle → Directive[Black ]] ,
   Shapiro Wilk == ShapiroWilkTest[tblSimulatedKthnnBin4[[All, 6]]],
   LabelStyle → Directive[FontFamily → "Times"]];

ShapiroWilkTest[tblSimulatedKthnnBin4[[All, 6]]]
(* Shaprio Wilk for Actual Data Bin 4 *)
0.12033

(* Normality of Bin 5 *)

plotSimhisttblKthnnBin5 = Histogram[tblSimulatedKthnnBin5[[All, 6]],
   ChartStyle -> White , Frame → True, FrameLabel → {"E", " "}] ;

plotSimqqtblKthnnBin5 = Labeled[
   QuantilePlot[tblSimulatedKthnnBin5[[All, 6]], PlotStyle → Directive[Black ] ],
   Shapiro Wilk == ShapiroWilkTest[tblSimulatedKthnnBin5[[All, 6]]],
   LabelStyle → Directive[FontFamily → "Times"]];

ShapiroWilkTest[tblSimulatedKthnnBin5[[All, 6]]]
(* Shaprio Wilk for Actual Data Bin 5 *)
0.0000556275
```

- Assemble and export the plot panel concerning normality of simulated data.

```
binImageCombinedSimulatedNormalityJpg =
 GraphicsGrid[{{plotSimhisttblKthnnBin1, plotSimqqtblKthnnBin1},
   {plotSimhisttblKthnnBin2, plotSimqqtblKthnnBin2}, {plotSimhisttblKthnnBin3,
    plotSimqqtblKthnnBin3}, {plotSimhisttblKthnnBin4, plotSimqqtblKthnnBin4},
   {plotSimhisttblKthnnBin5, plotSimqqtblKthnnBin5}}, ImageSize → {800, 800}]
```

E - Code for Mapping x,y Crater Positions to Lunar Latitude and Longitude.

```
#
# Stat 3080, R Group Spatial Randomness Crater Project
# Kurt Fisher and Russell Fuller
# Task 2 - Find plate solving equation and
#    transform coordinates from x,y to lunar lat long
# Version Date: 4-07-2013
# Version: 1.1
# File: 20130407PlateSolve1_1.R

# Uses: RandomPointSample.csv, Craters.csv
# Makes: CratersLatLong.csv - a file of x,y coords with their corresponding lat, long coords.

# Set your working directory.

# 1) Get plate coordinate data
options(digits=15)
obsamplePointsRaw = read.csv("RandomPointSample.csv", header=TRUE)
obsamplePointsRaw

obsamplePoints <- subset(obsamplePointsRaw, select=c("Id","x",
"y","Sample","Line","Planetocentric.Latitude","X360.Positive.East.Longitude","Local.Radius")) # just get the x,y columns
colnames(obsamplePoints) <- c("id","x","y","s","l","latpt","longpt","radiuspt") # Clean up names

# 2) Add an outlier column
col9 <- rep(0,length(obsamplePoints$x))
obsamplePoints <- cbind(obsamplePoints,isOt=col9)

# 3) Build a linear model to obtain plate solving equation for longitude
summary(lm1 <- lm(obsamplePoints$longpt ~ obsamplePoints$x + obsamplePoints$y, data = obsamplePoints,
direction="both")) # Make the model
slm1 <- step(lm1) # summarize the model
slm1Summary <- summary(slm1) # make the summary
summary(slm1) # display the summary
slm1$anova # check the df and residuals
plot(lm1, which=2)
plot(lm1, which=1)

# 4) Mark outliers
obsamplePoints[39,9] <- 1

# 5) Redo the longitude model without the outlier
summary(lm2 <- lm(obsamplePoints$longpt ~ obsamplePoints$x + obsamplePoints$y,
subset=(obsamplePoints$isOt==0), data = obsamplePoints, direction="both")) # Make the model
slm2 <- step(lm2) # summarize the model
slm2Summary <- summary(slm2) # make the summary
summary(slm2) # display the summary
slm2$anova # check the df and residuals
plot(lm2, which=1)

# 6) Redo the longitude model without the outlier y^2
summary(lm3 <- lm(obsamplePoints$longpt ~ obsamplePoints$x + obsamplePoints$y + I(obsamplePoints$y^2),
subset=(obsamplePoints$isOt==0), data = obsamplePoints, direction="both")) # Make the model
slm3 <- step(lm3) # summarize the model
slm3Summary <- summary(slm3) # make the summary
summary(slm3) # display the summary
slm3$anova # check the df and residuals
par(mfrow=c(2,2))
plot(lm3)
par(mfrow=c(1,1))
```

```
# 7) Check for influence points on the longitude model
library(car)
#   Make influence plot and save.
influencePlot(lm3)
#  Concluded to leave the influence point alone.
# Remove packages
detach("package:car", unload=TRUE)
detach("package:nnet", unload=TRUE)
# Usage Note:  Remember, after running this task, check that packages
# car, colorspace, effects, and nnet are disabled.

# 8) Conclusion lm2 with x-y is the best fit model for the longitude model.

# 9) Make the best-fit-model for longitude

funcBFMlong <- function(xhat,yhat)
{
  longhat <- (3.55506773145*(10^2)) + ((3.91987850137*(10^-5)) * xhat) + ((-4.64324471343*(10^-9)) * yhat)
  return(longhat)
}

# 10) Test the fit
plot(obsamplePoints$long,funcBFMlong(obsamplePoints$x,obsamplePoints$y))

# 11) Build a linear model for the latitude points - without the outlier
summary(lm4 <- lm(obsamplePoints$latpt ~ obsamplePoints$x + obsamplePoints$y,
subset=(obsamplePoints$isOt==0), data = obsamplePoints, direction="both")) # Make the model
slm4 <- step(lm4) # summarize the model
slm4Summary <- summary(slm4) # make the summary
summary(slm4) # display the summary
slm4$anova # check the df and residuals
plot(lm4, which=2)
plot(lm4, which=1)

# 12) Build a linear model for the latitude points - without the outlier
#  Note model omits x and use y^3 only.
summary(lm5 <- lm(obsamplePoints$latpt ~ obsamplePoints$y + I(obsamplePoints$y^2) + I(obsamplePoints$y^3),
subset=(obsamplePoints$isOt==0), data = obsamplePoints, direction="both")) # Make the model
slm5 <- step(lm5) # summarize the model
slm5Summary <- summary(slm4) # make the summary
summary(slm5) # display the summary
slm5$anova # check the df and residuals
plot(lm5, which=2)
plot(lm5, which=1)

# 13) Check for influence points on the longitude model
library(car)
#   Make influence plot and save.
influencePlot(lm5)
#  Concluded to leave the influence point alone.
# Remove packages
detach("package:car", unload=TRUE)
detach("package:nnet", unload=TRUE)
# Usage Note:  Remember, after running this task, check that packages
# car, colorspace, effects, and nnet are disabled.

# 14) Conclusion lm5 with y^3 is the best fit model for the latitude model.

# 15) Make the best-fit-model for longitude

funcBFMlat <- function(xhat,yhat)
{
```

```
  lathat <- (-3.25633152816*(10^1)) + (0 * xhat) + ((-3.28696665477*(10^-5)) * yhat) + ((-2.5658939655*(10^-12)) *
yhat^2) + ((1.68934247952*(10^-15)) * yhat^3)
  return(lathat)
}

# 16) Test the fit
plot(obsamplePoints$lat,funcBFMlat(obsamplePoints$x,obsamplePoints$y))

# 17) Get the all 890 observation points
craterData = read.csv("Craters.csv", header=TRUE)
names(craterData) # check the names
craterPoints <- subset(craterData, select=c("CraterNo", "X", "Y")) # just get the x,y columns
craterPoints <- craterPoints[c(-891,-892,-893,-894,-895),] # drop extraneous rows in source
colnames(craterPoints) <- c("CraterNo","x","y") # clean up the names
names(craterPoints)

# 18) Map the crater points to lat, long
col4 <- funcBFMlong(craterPoints$x,craterPoints$y)
col5 <- funcBFMlat(craterPoints$x,craterPoints$y)

# 19) Build a new output file
CratersLatLong <- cbind(craterPoints,long=col4,lat=col5)
names(CratersLatLong)

# 20) Save the output file
write.csv(CratersLatLong, file="CratersLatLong.csv")
```

[End]