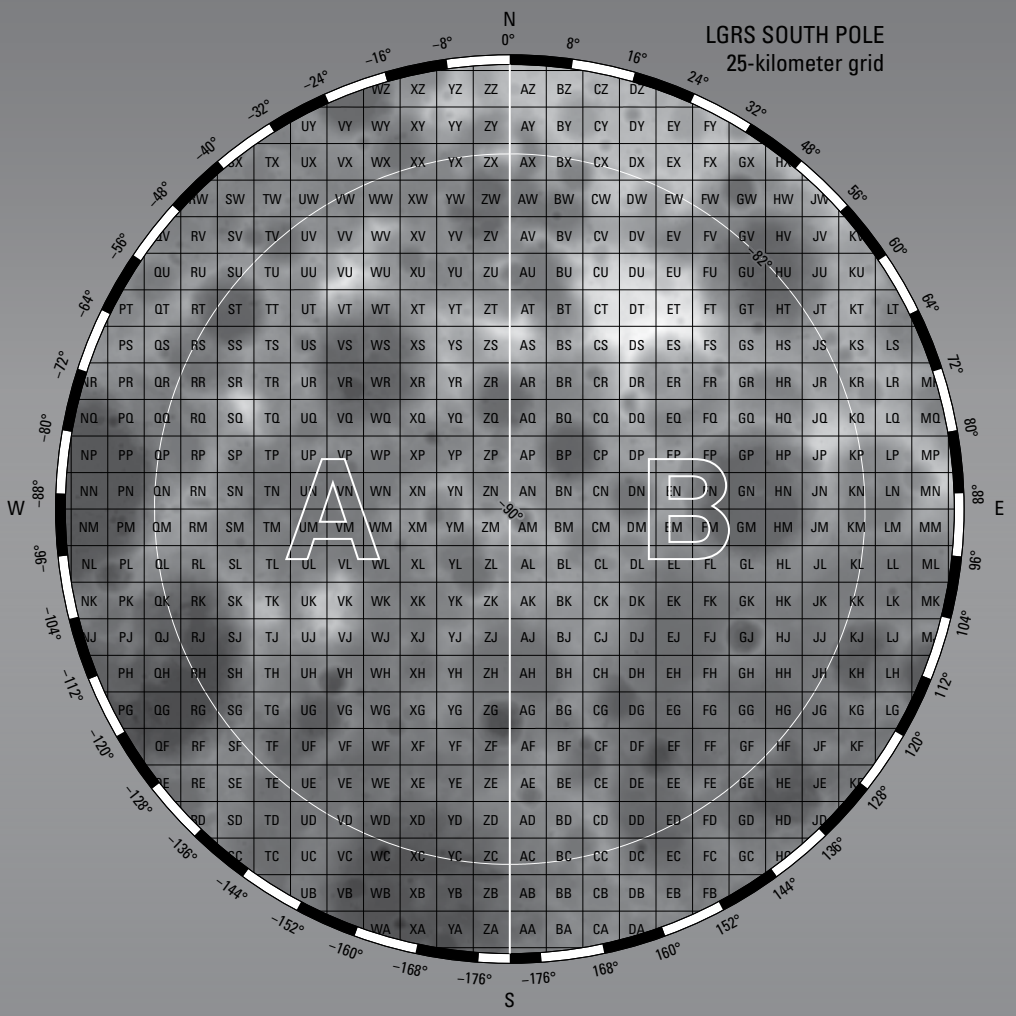
USGS
science for a changing world

Prepared in cooperation with the National Aeronautics and Space Administration and with support from the National Geodetic Survey

Lunar Grid Systems, Coordinate Systems, and Map Projections for the Artemis Missions and Lunar Surface Navigation

Chapter 1 of
Section E, Lunar Geodetic Standards
Book 11, Collection and Delineation of Spatial Data



Techniques and Methods 11–E1

U.S. Department of the Interior
U.S. Geological Survey

Front cover. Map showing south polar Lunar Grid Reference System global and 25-kilometer-grid areas and labels.

Back cover. Map showing Lunar Grid Reference System north polar region zones designated Y and Z and band-X zones in the extended Lunar Transverse Mercator range.

Lunar Grid Systems, Coordinate Systems, and Map Projections for the Artemis Missions and Lunar Surface Navigation

By Mark T. McClernan, Michael L. Dennis, Ike H. Theriot, Trent M. Hare, Brent A. Archinal, Lillian R. Ostrach, Marc A. Hunter, Matthew J. Miller, Ross A. Beyer, Andrew M. Annex, Samuel J. Lawrence

Chapter 1 of
Section E, Lunar Geodetic Standards
Book 11, Collection and Delineation of Spatial Data

Prepared in cooperation with the National Aeronautics and Space Administration and with support from the National Geodetic Survey

Techniques and Methods 11–E1

U.S. Department of the Interior
U.S. Geological Survey

U.S. Geological Survey, Reston, Virginia: 2025

Suggested citation:
McClernan, M.T., Dennis, M.L., Theriot, I.H., Hare, T.M., Archinal, B.A., Ostrach, L.R., Hunter, M.A., Miller, M.J., Beyer, R.A., Annex, A.M., and Lawrence, S.J., 2025, Lunar grid systems, coordinate systems, and map projections for the Artemis missions and lunar surface navigation: U.S. Geological Survey Techniques and Methods, book 11, chap. E1, 308 p., https://doi.org/10.3133/tm11E1.

Associated software release:
McClernan, M.T., Dennis, M.L., Theriot, I.H., Hare, T.M., Archinal, B.A., Ostrach, L.R., Hunter, M.A., Miller, M.J., Beyer, R.A., Annex, A.M., and Lawrence, S.J., 2024, Lunar map projections and grid reference system for Artemis astronaut surface navigation: U.S. Geological Survey, Astropedia Lunar and Planetary Cartographic Catalog, https://astrogeology.usgs.gov/search/map//lunar-map-projections-and-grid-reference-system-for-artemis-astronaut-surface-navigation.

ISSN 2328-7055 (online)

Foreword

This document contains design specifications of a navigational standard for the Moon, including a Lunar Transverse Mercator system, a Lunar Polar Stereographic system, a Lunar Grid Reference System, and a unique coordinate structure, Artemis Condensed Coordinates, for Artemis mission navigation and lunar surface science.

The National Aeronautics and Space Administration (NASA) Artemis campaign seeks to place humans on the Moon for the first time since the Apollo missions. Early Artemis missions are heavily focused on the lunar south pole, which promises to return valuable data on the Moon's geologic record, amongst other mission objectives. Coordinate systems in use today for the lunar south pole provides crew members on the surface neither an efficient nor intuitive means to communicate their position and orientation. A novel grid coordinate system, the Lunar Grid Reference System, is proposed to address these concerns for use in real-time extravehicular activity operations on the lunar surface.

The many stakeholders involved in the Artemis missions will need a common system to communicate position and orientation while astronauts are operating on the lunar surface. To that end, Artemis crew members will need that system to be *efficient* and *intuitive* to promote efficient extravehicular activity timelines and reduce confusion. In the context of this document, these characteristics are addressed on the design of lunar coordinate systems:

- *Efficient*.—The number of characters required to communicate a location within a desired precision level in both local and global contexts, and how many steps are required for a recipient or sender to interpret a location.
- *Intuitive*.—How well the system aligns with human perceptual abilities, and whether the system yields distances that have the same relationship to actual lunar surface distance in all directions from the point where a person is located.

Technological systems are currently being investigated to supplement the crew members' ability to locate and orient themselves and other assets on the lunar surface; however, it is unlikely that those systems will be fully operational for the first few landed missions. Even with future positional aids, crew members will still need an efficient and intuitive means to communicate position and orientation. In addition, if technological systems fail, the crews will require land navigation skills and have maps available, thus providing further motivation for a crew-centric coordinate system.

The contents of this U.S. Geological Survey (USGS) document detail a comprehensive framework for standardizing lunar crewed surface navigation within NASA and outlines the protocols, methods, and designs necessary for achieving consistency and interoperability across relevant space mission teams and lunar surface navigators. Key components of this document include designs of map projections, projected coordinate reference systems (Lunar Transverse Mercator and Lunar Polar Stereographic systems), and a grid system (Lunar Grid Reference System and Artemis Condensed Coordinates) for the Moon.

The work proposed in this document seeks to accomplish something similar to the National Geospatial-Intelligence Agency (NGA) document SIG 0012 (NGA, 2014a), but for using grid systems for the Moon. This report incorporates initial feedback and input from NASA's Artemis Geospatial Data Team, NASA's Flight Operations Directorate, National Geodetic Survey, USGS Astrogeology Science Center, and NGA and is intended to serve as a resource for all involved with the Artemis missions, as well as for engineers designing and operating lunar infrastructure.

Acknowledgments

The completion of this work was made possible through the collaboration and support of various individuals and organizations. The authors would like to express sincere gratitude to the National Aeronautics and Space Administration (NASA) Flight Operations Directorate, scientists, and staff at Johnson Space Center and in the Artemis Program who generously shared their expertise and resources, enhancing the quality and scope of our research.

Special thanks to the National Geodetic Survey, whose technical support and guidance on map projection design, map projection distortions, and the State Plane Coordinate System have led to the successful completion of this project.

Furthermore, we acknowledge the support and encouragement received from the National Geospatial-Intelligence Agency (NGA), which played a crucial role in shaping our work. We have appreciated the open and collaborative nature of NGA's role in geodesy and the public availability of their mapping standards, which were referenced and inspired the outcomes of this work.

Efforts from NASA's Johnson Space Center and the SETI Institute to review the initial software associated with this work and this Techniques and Methods chapter have been greatly appreciated.

We are grateful for the support this project received from the scientists on the Artemis Geospatial Data Team and look forward to future opportunities for joint endeavors to support astronauts and lunar exploration.

Lastly, we thank Regan Austin, Katie Sullivan, and Cory Hurd of the U.S. Geological Survey and anonymous reviewers for their assistance in the finalization of this work.

Contents

Figures

Tables

Abbreviations

ACC	Artemis Condensed Coordinates
AGDT	Artemis Geospatial Data Team
DMA	Defense Mapping Agency
DE	Development Ephemeris
DEM	digital elevation model
EPSG	European Petroleum Survey Group
EVA	extravehicular activity
GIS	geographic information system
GPS	Global Positioning System
GRAIL	Gravity Recovery and Interior Laboratory
IAG	International Association of Geodesy
IAU	International Astronomical Union
ICRF	International Celestial Reference Frame
JPL	Jet Propulsion Laboratory
JETT	Joint EVA Test Team
LDP	low-distortion projection
LGCWG	Lunar Geodesy and Cartography Working Group
LGRS	Lunar Grid Reference System
LLR	Lunar Laser Ranging
LOLA	Lunar Orbiter Laser Altimeter
LPS	Lunar Polar Stereographic
LRS	lunar reference system
LRO	Lunar Reconnaissance Orbiter
LROC	Lunar Reconnaissance Orbiter Camera
LTM	Lunar Transverse Mercator
ME	mean Earth system, lunar orientation
MGRS	Military Grid Reference System
NAD 83	North American Datum of 1983
NASA	National Aeronautics and Space Administration
NATO	North Atlantic Treaty Organization
NGA	National Geospatial-Intelligence Agency
NGS	National Geodetic Survey
OODA	observe, orient, decide and act
PA	principal axis
PCRS	projected coordinate reference system
PDS	Planetary Data System
PNT	positioning, navigation, and timing
TOD	true-of-date
USGS	U.S. Geological Survey
UPS	Universal Polar Stereographic
UTM	Universal Transverse Mercator
WG	working group
WGS84	World Geodetic System 1984
WKT	well-known text

Symbols

a	semi-major axis of the ellipsoid or spheroid
A	rectifying radius
α_{2k}	transverse Mercator forward coefficients
β_{2k}	transverse Mercator inverse coefficients
c	central angle between two points
$char$	unspecified character value
C_F	combined factor—from projection scale factor and height factor
d	horizontal distance between two points along the topographic surface
δ	map linear distortion—analogous to $C_F - 1$
e	eccentricity of an ellipsoid
E	grid easting coordinate
$E1k$	ACC 1-km-grid easting area designator
$E1k_{\text{num}}$	ACC 1-km-grid easting position in LPS or LTM
$E25k$	LGRS 25-km-grid easting area designator
$E25k_{\text{num}}$	LGRS 25-km-grid easting position in LPS or LTM
f	ellipsoidal flattening
F_E	false easting origin offset
F_N	false northing origin offset
G_d	distance between two points on a map grid
γ	grid scale convergence
H	hemisphere designator or geometric height
h	geometric height
h_F	height factor—map distortion from topography
i	grid coordinate reference index
k	grid (point) scale factor
$k - 1$	grid (point) scale error
k_0	central scale factor
λ	planetocentric longitude
λ_0	planetocentric longitude of projection origin
l	coordinate length
$latband$	LGRS 8° latitudinal band zone designator
$lonband$	LGRS LTM or LPS zone
M	coordinates in the mean earth axis orientation

η	transverse Mercator ratio Y/A
η'	Gauss-Schreiber ratio v/a
n	third flattening of the ellipsoid
N	grid northing coordinate
$N1k$	ACC 1-km-grid northing area designator
$N1k_{\text{num}}$	ACC 1-km-grid northing position in LPS or LTM
$N2M$	southernmost location of the 25-km northing letterset in LTM northings
$N25k$	LGRS 25-km-grid northing area designator
$N25k_{\text{num}}$	LGRS 25-km-grid northing position in LPS or LTM
N_{band}	LTM northing of the bottom of an LGRS 8° latitudinal band
φ	planetocentric latitude
φ_0	planetocentric latitude of projection origin
φ'	conformal latitude
φ_{band}	latitude of the bottom of an LGRS 8° latitudinal band
ω	longitude of difference $\lambda - \lambda_0$
p	variable in scale and convergence equations
p	grid precision using in coordinate truncation
P	coordinates in the principal axis orientation
P_1	vector used to convert between the ME and PA reference frames
P_2	vector used to convert between the ME and PA reference frames
q	variable in scale and convergence equations
ρ	planar distance from the center of the polar stereographic projection
$R_{\square}$	General rotation matrix. Subscript indicates rotation axis (*x*, *y*, or *z*)
σ	function of latitude
t	$t = \tan(\varphi)$
t'	$t' = \tan(\varphi')$
τ	angle used to convert between the ME and PA reference frames
θ	square of the half-chord length between two points
u	Gauss-Schreiber coordinate (north)
v	Gauss-Schreiber coordinate (east)
W	transverse Mercator grid-zone half-width
X	transverse Mercator *X* grid coordinate
ξ	transverse Mercator ratio X/A
ξ'	Gauss-Schreiber ratio u/v
Y	transverse Mercator *Y* grid coordinate
Z	grid system zone index

Lunar Grid Systems, Coordinate Systems, and Map Projections for the Artemis Missions and Lunar Surface Navigation

By Mark T. McClernan,[1] Michael L. Dennis,[2] Ike H. Theriot,[3] Trent M. Hare,[1] Brent A. Archinal,[1] Lillian R. Ostrach,[1] Marc A. Hunter,[1] Matthew J. Miller,[3,4] Ross A. Beyer,[3,5] Andrew M. Annex,[3,5] Samuel J. Lawrence[3]

Introduction

Lunar features are traditionally referenced from planetocentric coordinates of latitude and longitude in the mean Earth (ME)/polar axis system (Archinal and others, 2018). These coordinates can be spherical or Cartesian and satisfy virtually all lunar mapping and cartographic study needs as most lunar applications are remote sensing based. These planetocentric coordinates are based on angular measurements from the lunar equator, an adopted prime meridian that, on average, faces the Earth, and reference the Moon's center of mass (National Aeronautics and Space Administration [NASA], 2008). The use of these coordinates is widespread and fits current mapping needs; however, with NASA's planned return to the Moon during future Artemis missions, these coordinates are not well suited for use during active human navigation and are not intuitive for estimating distances.

At a fundamental level, the success of the Artemis mission objectives will be based on the Artemis mission astronauts' ability to navigate on the lunar surface. To date, a digital, modernized navigational system has not been implemented or optimized for human use on the Moon (McClernan and others, 2023). Near-term lunar navigation and future exploration (for example on Mars) may take place in austere environments lacking positioning, navigation, and timing (PNT) technologies and aids. Even in a PNT-enabled environment astronauts must have the capability to communicate position and orientation quickly and (or) in the event of equipment failure, ideally using a coordinate system designed using human performance guidelines. As such, the role of a standardized mapping system is highly important on the Moon given the challenges of navigating on another celestial body and the need for early Artemis missions to navigate using hard copy maps. Especially near the poles, a planar Cartesian coordinate system, that is rectangular (topocentric), should be used for navigation instead of a planetocentric coordinate system, since Cartesian coordinates can better maintain ground-based distances and orientation.

The need for a standardized coordinate system to navigate on the lunar surface was identified early in the Artemis Extravehicular Activity (EVA) program development and during the Joint EVA Test Team (JETT) analog field tests. Flight engineers from Johnson Space Center and the Artemis Geospatial Data Team (AGDT) requested that the U.S. Geological Survey's (USGS's) Astrogeology Science Center design a grid system for Artemis astronaut navigation (McClernan and others, 2023). This document includes designs for a navigation solution for astronaut surface travel. We have chosen to design a Cartesian grid system for our lunar navigational solution.

To support mission navigation, we have defined a globalized lunar navigation standard. Lunar surface navigation will be based on the rectangular, topocentric coordinate systems that consist of Lunar Transverse Mercator (LTM) and Lunar Polar Stereographic (LPS) projected coordinate reference systems (PCRSs). These systems are used to support a Lunar Grid Reference System (LGRS) and a unique grid system reporting structure, referred to as "Artemis Condensed Coordinates" (ACC), for mission navigation. The PCRSs and grid system specified in this document are more intuitive and better suited for the needs of active human navigation than planetocentric latitude and longitude coordinates. The LTM and LPS systems and the LGRS are similar in design to the terrestrial Universal Transverse Mercator (UTM) and Universal Polar Stereographic (UPS) systems and the Military Grid Reference System (MGRS) (National Geospatial-Intelligence Agency [NGA], 2014b). However, the systems designed in this report differ in that they are designed for NASA's specific use on the Moon. ACC is a conversion for LGRS coordinates that produces a coordinate format similar to historic Army Map Service topographic orthophoto charts and the crew-centric grid system used for Apollo mission lunar navigation (NASA, 1969, 1973).

[1]U.S. Geological Survey

[2]National Oceanic and Atmospheric Administration

[3]National Aeronautics and Space Administration

[4]Jacobs Technology, Inc.

[5]SETI Institute

ACC was designed specifically to limit the number of unique characters required to communicate a position estimate on the lunar surface during Artemis missions. The LTM and LPS systems and the LGRS are general navigational systems, and each can be utilized as a standalone system. ACC condenses LGRS coordinates to report relative position with only alphanumeric characters to a desired level of precision. ACC cannot be used as a standalone grid system as it is based on LGRS.

The LTM and LPS systems and the LGRS are described in this document. They are intended to serve as an Artemis mission engineering, surveying, and navigational solution for astronauts' use on the lunar surface.

Artemis Mission Needs

In early 2023, NASA's AGDT requested the USGS's assistance with Artemis missions through the creation of controlled elevation models and orthorectified imagery, the verification and validation of general spatial data products, and the creation of a grid system for mission navigation (McClernan and others, 2023). The USGS also has personnel involved with teaching geologic mapping during the JETT astronaut training as well as archiving lunar data as part of the NASA Planetary Data System (PDS) Cartography and Imaging Sciences Discipline Node. The AGDT was created as a common work environment between NASA and other partners to track the geospatial needs of the Artemis missions. The lunar navigation standard has been an ongoing project within the AGDT.

Grid System for Navigation

The Artemis navigational grid is designed to support a globalized system for human navigation that is capable of being communicated with a limited number of characters. It adheres to the following technical recommendations and requirements provided by NASA's Flight Operations Directorate to the AGDT (McClernan and others, 2023):

- Requirement 1.—Navigational grids need to work for the entire Moon, including: polar, equatorial, and global use cases.
- Requirement 2.—The grid must utilize a coordinates scheme that has an even number of characters, limited to six characters when reporting relative position.
- Requirement 3.—The coordinates must communicate position accurately to a 10 meter (m) × 10 m precision globally.
- Recommendation 1.—The projections utilized will ideally have a maximum scale error of approximately ±1:1,000 (1 m per kilometer [km]).

Using LTM and LPS PCRSs together in the LGRS creates a global system that meets requirement 1. Condensing the LGRS coordinates using ACC addresses requirements 2 and 3. We discuss the PCRSs in the "Transverse Mercator and Lunar Transverse Mercator (LTM) System" and "Polar Stereographic and Lunar Polar Stereographic (LPS) System" sections and grids in the "Lunar Grid Systems" section.

Recommendation 1, which specifies a tolerance for map projection scale error, restricts the allowable range for the central scale factor. The scale parameter helps to quantify how well distances and dimensions on a map match the actual distances on the topographic surface. Although maps with scale errors within the specified ±1:1,000 tolerance are possible on the Moon, these map projections are impractical for global or even polar coverage, because they would require many zones, each covering a small area of the lunar surface. This information was communicated to NASA's Flight Operations Directorate, and they accepted this larger scale error to finalize the grid system (Flight Operations Directorate , NASA, oral commun., 2023). To achieve a scale error of ±1:1,000 or less, low-distortion projections (LDPs) can be designed and utilized, similar to the LDPs designed for the new State Plane Coordinate System (Dennis, 2023); however, that has not been done for the Moon in this document.

Background

Establishment of robust lunar geodetic standards is paramount for accurate spatial referencing and scientific analysis. Such geodetic standards serve as a foundational framework for precisely measuring and mapping the surface features of the Moon. A defined lunar reference system (LRS) exists (Davies and others, 1980; NASA, 2008; Folkner and others, 2008; Archinal and others, 2011, 2018), which includes a set of well-defined reference points including a reference frame, a (spherical) reference surface, and geodetic coordinate systems. This system supports existing measurement techniques that provide mission planners with consistent and reproducible positioning, and it enables reliable comparison of data collected from diverse lunar missions and instruments. We do not define a new LRS in this work but utilize the currently accepted LRS to support the PCRSs and grid system defined in this report.

This section introduces the current technical background behind map projection design, imposed distortions, grid systems, and the current internationally accepted geodetic standards for the Moon.

Projected Coordinate Reference Systems (PCRSs)

Geospatial information can be represented in a three-dimensional (3D) coordinate system that represents position as a geocentric coordinate triplicate of latitude, longitude, and height with respect to the spheroidal model of the Earth or other celestial bodies (NASA, 2008; Folkner and others, 2008; Archinal and others, 2011, 2018). These positions are based on horizontal frames, such as the World Geodetic System 1984 (WGS84) or the North American Datum of 1983 (NAD 83) (NGA, 2014a, and Snay, 2012, respectively). Height can be represented and determined in a variety of ways, such as geopotential-based heights with respect to the geoid (NGA, 2014a) or geometric heights such as WGS84 ellipsoidal heights (NGA, 2014a). A spatial reference system and associated modeling is beyond the scope of this document. Instead, we discuss development of PCRSs from the context of an existing system that defines latitude, longitude, and height.

In addition to a PCRS defining a map projection, it must also use a specific reference frame realization (for example, the WGS 84 (G2296) or NAD 83 (National Adjustment of 2011) frames for the Earth), which must include a specific reference ellipsoid (WGS 84 and GRS 80, respectively, for this example). The same is true for a lunar PCRS, to ensure that it defines the correct location on the Moon. Although any lunar reference frame can be used if proper transformations are applied, it is essential that the reference frame of a PCRS be correctly and consistently defined to avoid positioning errors.

In human navigation, positioning coordinates are often converted to a more practical format from latitude and longitude. To facilitate navigation, a local Cartesian system is used, typically as a topocentric ("rectangular") coordinate system, expressed in a coordinate of eastings and northings, (Canters, 2002; Markič and others, 2018; Dennis, 2018); geometric height can be included as a third dimension. Topocentric coordinates are expressed in a rectangular frame with an east and a north axis that can be conceptualized as forming a local plane parallel to the planetary reference surface at a point of origin (Snyder, 1987; Canters, 2002; Markič and others, 2018; Dennis, 2023). PCRSs are mathematical conversions that rigorously relate topocentric coordinates to the reference surface and are fundamental to accurately representing a 3D celestial body onto two-dimensional (2D) maps (Snyder, 1987; Dennis, 2022). Conformal map projections are commonly used in PCRSs for navigation, which locally preserves angular orientation and are the only type of projection where the scale error is the same in every direction for all points. For navigation, commonly used conformal map projections include the Lambert conformal conic, regular Mercator, transverse Mercator, stereographic, and Hotine oblique Mercator—transverse Mercator and polar stereographic are considered in this report. Conformal map projections generally do not preserve area and (like all other projections) can have large scale error (that is, linear distortion, δ) away from the projection axis or in areas of significant topographic relief (projection axis defined below). Thus, the useful coverage area of these (and all other) projections can be limited owing to excessive distortion, depending on the application.

For the LGRS, designing a PCRS is limited to conformal projections, with a focus on transverse Mercator and stereographic projections. Common map projection design parameters are as follows:

- **Map projection type.**—The projection type is typically selected based on the shape and location of an area (although other considerations, such as topographic slope, can be relevant if designing LDPs). For example, transverse Mercator projection is often used for areas that are long in the north-south direction, and stereographic projection is often used in regions centered on the poles.

- **Projection axis location.**—The projection axis is a line or point where the scale factor is at its minimum value and (usually) constant. It is defined by a specific latitude and (or) longitude, $\{\varphi_0, \lambda_0\}$, where $\{\lambda_0\}$ is the central meridian of the transverse Mercator projection, and $\{\varphi_0, \lambda_0\}$ is the origin of the stereographic projection. Typically, it is centered on the area where the map projection will be used.

- **Central scale factor k_0.**—The central scale factor quantifies how much the map projection axis is scaled relative to the reference surface (ellipsoid or sphere). It is usually selected such that the negative and positive scale error is balanced over the coverage area of the PCRS and the overall scale error is near zero.

- **Grid origin.**—The grid origin is the latitude and longitude where the false easting and northing $\{F_E, F_N\}$ are specified. Usually, these values are selected such that the easting and northing are positive everywhere within the PCRS coverage area. Here we use the same latitude and longitude for the grid origin and the projection axis location, $\{\varphi_0, \lambda_0\}$ (although that is not always the case), and transverse Mercator projection axis does not include $\{\varphi_0\}$, even though it is needed for the grid origin.

- **Reference frame and linear unit.**—To make a PCRS complete and usable in practice, it must include a reference frame (also known as a datum of a geographic coordinate system) and a specified linear unit.

- **Other parameters.**—Other projection types require additional parameters, such as standard parallels for the two-parallel Lambert conformal conic projection or the skew-axis azimuth of the oblique Mercator projection. However, these parameters are not needed for the projection types discussed in this document.

By combining these parameters, a PCRS can be defined that minimizes linear distortion within the coverage area. The definition and calculation of map projection distortion is discussed in the next section.

Map Projection Linear Distortions

Map projection linear distortion quantifies the difference between an actual distance on the curved topographic surface of the planetary body and its representation on the mapping plane ("grid"); it is often considered equivalent to the grid scale factor, k (Snyder, 1987; Stem, 1990; Canters, 2002; McClernan and others, 2023). However, a distinction is drawn here between the two terms: linear distortion is determined at the topographic surface ("ground"), whereas the scale factor is determined at the surface of the reference ellipsoid or sphere, as is the scale error ($k - 1$). The difference between "grid" and "ground" distance caused by linear distortion can be problematic for many geospatial products where a high degree of spatial accuracy is required. Such distortion is a consequence of converting the Moon's curvature and irregular topographic surface to a flat map. Although linear distortion cannot be eliminated, its effect can be minimized.

Many map projections used for navigation, surveying, engineering, and construction are conformal, meaning that linear distortion at a point does not vary with orientation (Snyder, 1987; Stem, 1990; Canters, 2002; Dennis, 2016; McClernan, and others, 2023). Angular distortion also does not vary with direction and is given as a constant convergence (mapping) angle at a point, which is the difference between true north and grid north (Snyder, 1987). While angular direction (and thus shape) is preserved locally at a point, there is generally distortion in distance.

Although linear distortion can be considered in terms of actual physical distances, it is defined at a point for an infinitesimal distance (Snyder, 1987; Stem, 1990; Canters, 2002; Dennis, 2016). As mentioned above, scale error is commonly specified with respect to the reference surface. However, distortion at the topographic surface is often of great interest since that is where most human activity occurs. The associated distortion and its effect are of critical importance to calculate distances correctly on the Moon, given its large topographic variation (Lunar Reconnaissance Orbiter Camera [LROC] Team, 2013). To determine linear distortion, we first calculate the combined factor, C_F (from Stem, 1990):

$$C_F = k \times h_F, \tag{1}$$

where

k is the grid scale factor, and
h_F is the height factor (Stem, 1990; Canters, 2002; Dennis, 2016).

Historically, h_F is referred to as the "elevation factor" or "sea level factor" (Stem, 1990).
The grid scale factor is fully described in later sections, as it is computed differently for different map projections. The height factor, h_F, is defined as

$$h_F = \frac{a}{a+h}, \tag{2}$$

where

a is the reference surface radius of the celestial body, and
h is the geometric height of terrain above or below the reference surface used to account for the effect of topographic relief (Stem, 1990; Canters, 2002; Dennis, 2022).

The use of a radial distance is unique to reference frames that use a sphere, such as the Moon, because curvature is constant. When a reference ellipsoid is used, such as for the Earth, curvature of the reference surface varies with direction owing to flattening. In those cases, a different parameter is used that accounts for that variation, such as the geometric mean (or Gaussian) radius of curvature (Dennis, 2016, 2022, 2023).

Linear distortion is commonly defined as δ, which is analogous to scale error, $k - 1$ (for example, Dennis 2016, 2022, 2023). δ expresses distortion as a ratio between map or "grid" distance to actual or "ground" distance, for example in parts per million or as a scale. This is useful for evaluating PCRSs with low distortion, such as LDPs. In this report, $\delta = C_F - 1$ and is similarly defined as linear distortion.

Linear distortion can be expressed as a unitless ratio of distorted distance to true distance and is typically a scale or decimal value near 0. For example, $C_F = 0.999$ represents a linear distortion of −1:1,000, a minus-one-part-per-thousand or a minus-one-meter-per-kilometer (m/km), which means that the projected grid distance is 1 m shorter than a 1 km horizontal (curved) ground distance. Similarly, $C_F = 1.001$ represents a linear distortion of +1:1,000 (+1 m/km), meaning that a grid distance is 1 m *longer* than a 1 km ground distance. Thus, if , $C_F > 1$ grid distances are longer than ground distances, whereas if $C_F < 1$, grid distances are shorter than ground distances, and if $C_F = 1$, grid and ground distances are not distorted. Linear distortion in meters per kilometer can be calculated from C_F

$$\delta_{\mathrm{m/km}} = \delta \times 1{,}000 = (C_F - 1) \times 1{,}000. \tag{3}$$

Although distortion is technically defined at a point, it is often desirable to consider it over a finite distance, such as a line between two map-projected points. Such distance is nearly always distorted (that is, shorter or longer than the actual horizontal ground distance) but can be corrected to estimate the ground distance, d, using the following equation (modified from Stem, 1990):

$$d = \frac{G_d}{\overline{C_F}}, \tag{4}$$

where

G_d is the grid distance, and

$\overline{C_F}$ is the mean combined factor at the end points of the line.

This provides an estimate of the horizontal ground distance because the grid scale factor varies continuously along the line. However, for point pairs located within a few hundred kilometers of the projection axis, this estimate is accurate to better than 1 millimeter (mm) for distances of less than 10 km on Earth (Dennis, 2022). On the Moon, the distance that can be corrected to better than 1-mm accuracy would be somewhat less (owing to its smaller size and thus greater curvature), but it would still be several kilometers, which is sufficient for most practical applications. For longer distances or greater accuracy, it may be necessary to numerically integrate the grid scale factor along the line (Stem, 1990, p. 49–50; Dennis, 2022, p. 43–44). Other methods that do not rely on map projections are available for accurately computing long ground distances (see Dennis, 2022, p. 58–61).

Linear distortion and its variation over a map area can be minimized by selecting an appropriate map projection and minimizing the linear distortion magnitude, $|C_F - 1|$. For maps covering large areas, the effect of topography can be ignored, since total distortion becomes strongly dominated by distortion due to curvature. This occurs because distortion due to curvature increases with approximately the square of distance from the projection axis, whereas distortion due to change in topographic height increases linearly (Dennis, 2022). In such cases it may be sufficient to minimize only the scale error magnitude, $|k - 1|$. Owing to NASA's interest in total distortion (that is, at the topographic surface), we have extended our PCRS scale-error analysis calculations to include the effect of topography using the combined factor, eq. 1, in addition to using only the grid scale factor (that is, at the surface of the lunar reference sphere). The analysis includes relevant plots of the grid scale factor, the height factor, and the combined factor in sections of this document that discuss the projection areas of the LTM and LPS systems.

Grid System Heritage

This section discusses grid systems such as the Military Grid Reference System (MGRS) and the grid systems used for each Apollo mission. This heritage can be used for the grid systems for Artemis missions and the current and ongoing JETT analog testing.

Military Grid Reference System Heritage

Local grid reference systems have been critical to military operations since the deployment of long-range artillery, most notably during World War I, to rapidly locate targets, calculate bearing and distance between positions, and coordinate efforts across the battlefield. Each reference grid was developed to meet the operational usage requirements of the individual military unit in their area of operations, without consideration for other combat arms (Evans, 2014). Soon, the complexity of coordinating the indirect fire from several sources utilizing different coordinate systems created significant problems for commanders. The "Study and Discussion of Military Grids" analyzes the use of different reference grids by the British coastal artillery, Navy, and Air Force: "Intolerable confusion which resulted from the use of these grids during the numerous German air raids in the Battle of Britain makes it highly probable that these conflicting systems would have led to at least a few local disasters had there been an invasion of Great Britain" (Army Map Service and Military Intelligence Division, circa early 1945). Following World War II, the need for a global system that was adequate for all combat arms was recognized, and similar efforts were initiated by other militaries and governments.

Based on minimum operational requirements, a set of mapping specifications resulted in what would become known as MGRS (NGA, 2014b). In 1949 MGRS was implemented across the Department of Defense and was later adopted as the standard for all North Atlantic Treaty Organization (NATO) militaries. Since that time, MGRS has been key geodetic infrastructure for joint military operations, as well as civilian search and rescue, and topological scientific investigations (Palombo, 2021). MGRS is especially useful in bare environments lacking built infrastructure with known locations to reference. Additionally, the common standard has enabled direct coordination and exchange with very diverse personnel, enabling training in different areas across the globe.

Beyond the coordination of supporting arms in combat, MGRS has proven useful for a wide range of operations, such as mission planning, supporting logistics, and engineering tasks. A key component of MGRS that makes it broadly useful is the scalable precision within the same notation framework. For example, a full 10-digit MGRS grid coordinate references a single 1 m × 1 m area, readily provided by modern Global Positioning System (GPS) instruments, which may be used to locate a discrete threat or resource (for example, an explosive device identified by handheld metal detector). This same resolution, however, is not required for all supporting arms or casualty evacuations when read directly from a map. In such emergency situations, reduced resolution to 10 m × 10 m (eight-digit coordinate) or 100 m × 100 m (six-digit coordinate) areas may be equally useful while expediting the communication process.

Despite the difference in mission goals, there exist significant commonalities between military combat and lunar surface operations. Challenging physical environments, limited means of communication, and time-sensitive, high-stress tasks all place a burden on humans both on the ground and in mission control. The ability for a group to iterate through the observe, orient, decide, and act (OODA) loop in near real-time is essential for the successful execution of a task, equally important for both the military and for Artemis crewed surface operations.

Apollo Flight Heritage

Limited Apollo-era documentation that exists today regarding grid systems includes operational map books, mission transcripts, some scanned Army and Air Force training maps (NASA, 1969, 1973), and a paper by Kinsler (1975) discussing Defense Mapping Agency (DMA) map projections for Apollo-era lunar charting. Artemis flight engineers have compared the Apollo map books to the mission logs to evaluate them for any relevant parameters that can be utilized today.

Maps developed for the Apollo missions delineated operational areas for each mission (the entire areas the crews were reasonably able to reach given the target landing site) and created highly localized grid systems for the crews to reference. Various map parameters evolved throughout the Apollo campaign, as shown in table 1. Scales of the maps interpreted to be the primary maps for crew navigational use are noted.

Notably, the grid denomination grew as traverse distance and speed increased owing to the addition of the Lunar Roving Vehicle, and map scales were accordingly adjusted. Grid naming conventions were also updated, and most Apollo missions omitted the letters I and O, presumably to prevent confusion with the numbers 1 and 0. A lunar grid system like LGRS can be used at different map scales as movement speeds increase; it is expected that a coarse grid resolution will be needed in later Artemis missions.

Heritage Concerning Lunar Grid System for Artemis

NASA Johnson Space Center and USGS chose a grid system for Artemis mission navigation given the success of flight-tested grids used during the Apollo Program and the worldwide use of UTM, UPS, and MGRS (NGA, 2014b). However, directly repurposing MGRS

Table 1. Apollo mission maps, map scales, grid dimensions, and feature information.

[Compiled from Apollo 11–17 map books from NASA (1969) and Jones (2017). # indicates use of a character positions in a grid area label, * indicates the spacing the grid label represents, × delineates grid spacing where grid easting precedes northing, that is, *E*×*N*. Max., maximum; m, meter; km, kilometer; km², square kilometer; k, thousand; NSEW, north south east west]

Parameter	Apollo 12	Apollo 13/14	Apollo 15	Apollo 17
Grid interval	50 m	50 m	250 m	200 m
Naming convention	A–W (I/O omit) #–## (Each map had some grid labels)	AA–ZZ #–##	AA–ZZ (I/O omit) #–##	AA–ZZ (I/O omit) #–##
Max. definable operational area per convention	1 km × 1.3 km = 1.3 km²	33.8 km × #*50 m = 1,142.44 km²	144 km × #*250 m = 20,736 km²	115.2 km × #*200 m = 13,271 km²
Map/imagery scales provided to crew on the surface	1:5k [1] 1:25k 1:100k	1:5k [1] 1:25k 1:100k	1:5k (walking) 1:25k (rover) [1] 1:50k 1:100k 1:250k	1:5k 1:12.5k 1:25k (rover) [1] 1:50k 1:100k 1:250k
Max. possible resolution via voice only	5 m	5 m	25 m	20 m
Other features	Geologic units (unclear)	(Could not be determined)	Geologic overprint	Geologic units, contours, major crater outlines, horizon profiles, NSEW sun azimuth and altitude

[1]Maps used for crew surface navigation.

for the Moon is not ideal owing to the size difference between the Moon and the Earth. Figure 1 shows MGRS 100-km grid areas as dimensioned on the south pole of the Moon. As is shown, only 44 grid areas are utilized on the Moon to cover the south polar region south of lat −80°, whereas on Earth there are 432 100-km south polar MGRS grid areas. The limited number of grid areas of a 100-km grid is too coarse to uniquely identify areas near the south pole given the smaller size of the Moon (fig. 1). Additionally, initial EVAs are likely to be restricted to within a few kilometers of the Artemis landers; therefore, a finer grid is required, of similar scale to the Apollo EVA grids (table 1). As such, direct use of MGRS is not ideal as LGRS was designed to support a grid smaller than 100 km to NASA's specifications. A 25-km grid, as planned by LGRS, allows for the construction of 516 unique zones across the south pole of the Moon. This is closer to the number of 100-km MGRS grid areas for the south pole of the Earth while maintaining a consistent grid across the lunar surface.

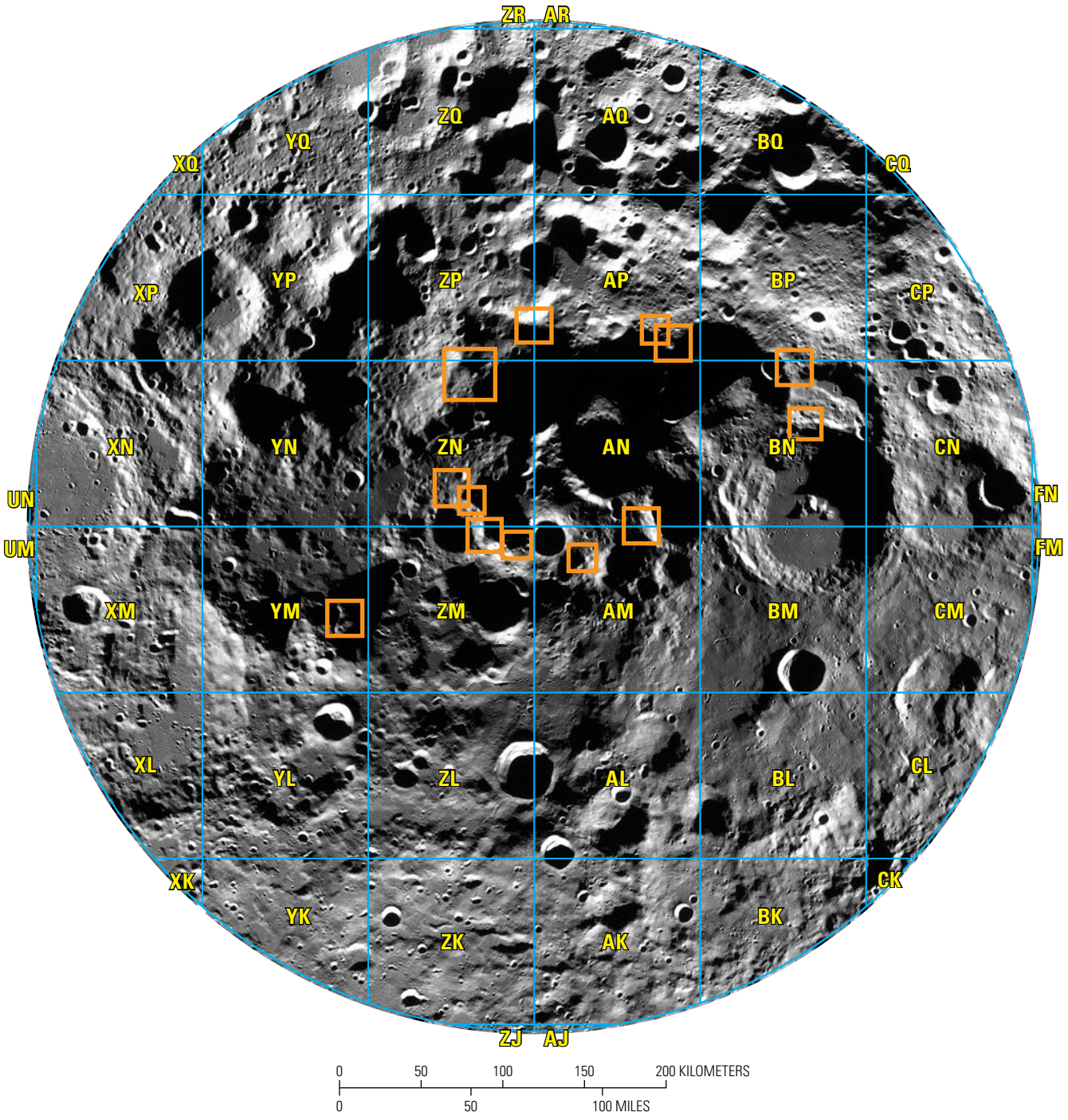


Figure 1. Military Grid Reference System (MGRS) 100-kilometer grid areas plotted on the south pole of the Moon (see National Geospatial-Intelligence Agency, 2014b). Artemis III candidate landing regions are outlined in orange. Image mosaic from Lunar Reconnaissance Orbiter Camera (LROC) Team (2013).

Map projections are dimensioned over a sphere over a specified angular distance (Snyder, 1987). As the Moon is a smaller celestial body, the surface distance a map projection can cover is greatly reduced. For instance, at each equator, a UTM zone that is 6° longitude wide spans 667 km on Earth, whereas it spans 181 km on the Moon. Moreover, at lat 80° north or south, 6° of longitude spans 116 km on Earth, whereas it spans 32 km on the Moon. Considering the physical implications, direct use of 6°-wide UTM zones is not practical owing to the smaller range of coverage at the pole and the increased number of zone transfers while navigating in these regions. LTM utilizes 8°-wide transverse Mercator zones (243 km at the lunar equator) to increase the surface distance of zones near the poles (spanning 42 km at 80° latitude), thus decreasing the number of potential zone transfers.

The polar stereographic projection utilized by LPS in this document is implemented similarly to the UPS system (Snyder, 1987), and although covering a smaller range, is not modified and is used in high-latitude regions north of lat 80° and south of lat −80°.

Comments on the JETT Astronaut Training Map Grids

The need for a grid coordinate system arose early in the Artemis EVA development process through the recognized need for the astronauts and ground teams to effectively communicate position and orientation data. Ground navigators typically use a north–south–east–west cardinal direction frame to keep them oriented in the terrain; however, due to meridian convergence at the poles, relative orientation to a cardinal direction becomes ambiguous. Various iterations of grid coordinate systems have been deployed in field testing since 2020 and have varied the grid denomination (50 m and 100 m), map scales (between 1:2,500 and 1:25,000), and grid naming convention to gather feedback on their use. In coordination with the USGS, NASA deployed the latest grid iteration for JETT Field Test 5 in May 2024, which utilized a 100-m grid and nomenclature based on the ACC structure proposed by the USGS and described in this document.

Current Lunar Reference System

The International Astronomical Union (IAU) Working Group (WG) for Cartographic Coordinates and Rotational Elements is responsible for recommending international standards for planetary geodesy including rotation rates, spin axes, prime meridians, and reference surfaces for individual planets and satellites, including the correct lunar reference frame and lunar reference surface (Davies, and others, 1980; Archinal and others, 2011, 2018). We use the current recommendations from the IAU WG (Davies, and others, 1980; Archinal and others, 2011, 2018) as defined by NASA (2008).

Current Lunar Standards and Similar Works

The current international standard, defined by Archinal and others (2018) is to use the lunar ME reference frame defined from the Jet Propulsion Laboratory (JPL) Development Ephemeris (DE) DE421 (Folkner and others, 2008). NASA and the USGS have generally followed IAU WG recommendations. In 2008, a white paper published by the Lunar Reconnaissance Orbiter (LRO) Project and the NASA-sponsored international Lunar Geodesy and Cartography Working Group (LGCWG) (LRO mission and LGCWG, 2008) provided a standardized coordinate system for the Moon, used here.

Lunar Reference Frames

A reference frame is fundamentally a model that defines lunar surface fixed coordinates (LRO mission and LGCWG, 2008; Archinal and others, 2011, 2018; Song, and others, 2021). Converting to a lunar reference frame starts from the International Celestial Reference Frame (ICRF), an inertial system independent of an epoch and based on distant extragalactic objects, fixed in position relative to the Moon as it rotates (Ma and others, 1998; Song and others, 2021; JPL, 2023). ICRF is the current recommendation for celestial reference systems and is adopted by the IAU (Archinal and others, 2011, 2018).

The ICRF then undergoes a series of sequential transformations to produce one of many desired target lunar reference frames (LRO mission and LGCWG, 2008; Archinal and others, 2011, 2018; Song and others, 2021), that is, a mean dynamical frame, a principal axis (PA) frame, or a mission specific reference frame such as the true-of-date (TOD) frame for the Korean Pathfinder Lunar Orbiter (Song and others, 2021).

The inertial reference frame of the Moon, sometimes referred to as the "mean dynamical frame of J2000.0 (Julian epoch)," is the ICRF centered on the Moon's center of mass, and used for processing observations involving planetary motion, such as Lunar Laser Ranging (LLR) (Archinal and others, 2011, 2018; Song and others, 2021). The ICRF term is interchangeable with the mean dynamical frame of J2000.0 despite a small difference of 0.1 arc-seconds (Ma and others, 1998; Archinal and others, 2011, 2018; Song and others, 2021). The inertial frame is defined from the direction of the north pole, a *z*-axis of the mean lunar rotation, and is specified using right ascension and declination with respect to the ICRF (LRO mission and LGCWG, 2008; Folkner and others, 2008; Archinal and others, 2011, 2018; Park and others, 2021; Song and others, 2021). The *x*-axis is oriented in the intersection of the equatorial and ecliptic planes, and the *y*-axis is rotated 90 degrees from the *x*-axis on the ecliptic plane. The ICRF is transformed into one of two body-fixed frames, ME or PA (Archinal and others, 2011, 2018; Song and others, 2021). These lunar body-fixed frames are referred to as "body-fixed-body-centered," a non-inertial orientation frame of reference where the origin is placed at the Moon's center of mass, and the frame is fixed in orientation with respect to the Moon's surface (ME) or axis of figure or gravity field (PA) as it rotates (LRO mission and LGCWG, 2008; Folkner and others, 2008; Archinal and others, 2011, 2018; Park and others, 2021; Song and others, 2021). LLR solutions are used to model the JPL DE. The DE is used to rotate the Moon from an inertial reference space (ICRF or dynamic frame of J2000.0) to a body-fixed system, ME or PA (LRO mission and LGCWG, 2008; Folkner and others, 2008; Park and others, 2021; Song and others, 2021). There are two slightly different lunar body-centered-body-fixed systems and their associated frames: PA, which is aligned with the principal axes of the Moon; and ME, which is based upon a mean direction to the Earth and the mean rotation axis of the Moon (Folkner and others, 2008; Archinal and others, 2011, 2018; Park and others, 2021). To generate coordinates in a PA frame, three Euler angles representing the physical lunar librations and a general rotation axis are used to represent position in the *z*- and *x*-axes (Folkner and others, 2008; Archinal and others, 2011, 2018; Park and others, 2021). The ME frame is based on a set of rotations from PA and are defined in the JPL DE documentation (Folkner and others, 2008; Park and others, 2021; JPL, 2023). For example, the generic transformation from PA to ME is defined using the following equation from Folkner and others (2008) and Park and others (2021):

$$M = R_x\left(-P_2\right) R_y\left(P_1\right) R_z\left(-\tau\right) P, \tag{5}$$

where

M and P are Cartesian points in the ME and PA reference frames,
P_1, P_2, and τ are the constant angles from DE in arcseconds, and
R is a first-order rotation matrix.

For the DE421 (Folkner and others, 2008),

$$M = R_x\left(-0.30''\right) R_y\left(-78.56''\right) R_z\left(-67.92''\right) P, \tag{6}$$

which has the inverse function of

$$P = R_x\left(67.92''\right) R_y\left(78.56''\right) R_z\left(0.30''\right) M\ . \tag{7}$$

For DE440 (Park and others, 2021),

$$M = R_x\left(-0.2785''\right) R_y\left(-78.6944''\right) R_z\left(-67.8526''\right) P, \tag{8}$$

which has the inverse function of

$$P = R_x\left(67.8526''\right) R_y\left(78.6944''\right) R_z\left(0.2785''\right) M\ . \tag{9}$$

ME and PA are not coincident, as a positional maximum difference of approximately 875 m is expressed on the lunar surface (LRO mission and LGCWG, 2008; Folkner and others, 2008; Park and others, 2021). Because of the Moon's spin and orbit, the PA frame *z*-axis does not coincide with the Moon's mean spin axis, nor does the *x*-axis coincide with the mean direction to the center of the Earth (JPL, 2023), but rather they are aligned to the principal axes of the body of the Moon. However, this is not the case in the ME frame where the orientation is now aligned with the previous realization of the ME frame and therefore the lunar surface. This also still approximately maintains the ME system orientation, of having the *z*-axis aligned with the north lunar rotation axis and the prime meridian facing the mean Earth direction (JPL, 2023). The procedure for updating the orientation of the ME frame is consistent with how the International Terrestrial Reference Frame is updated for the Earth. ME is also used to describe the orientation of the Moon relative to the ICRF and is the current PDS-required lunar frame for data archiving (JPL, 2009). Other reference frames exist such as the Moon TOD frame or other historic frames (Song and others, 2021). However, they are beyond the scope of this document.

Lunar Reference Surface

Separate from the orientation model of the reference frame is a lunar reference surface, a geometric shape of the Moon for cartographic use (Archinal and others, 2011, 2018). The scientific community has converged on the use of a sphere with a radius of 1,737,400 m (LRO mission and LGCWG, 2008; Folkner and others, 2008; Archinal and others, 2011, 2018; Park and others, 2021; Song and others, 2021). The general shape of the Moon is a sphere, excluding local topography variations (JPL, 2005). Flattening of this surface does exist; however, the IAU and the International Association of Geodesy (IAG) have recommended using the same equatorial and polar radius values for any mission design and navigation analyses (JPL, 2005; Archinal and others, 2011, 2018). A spheroid for the lunar surface was also selected for lunar topography data, as lunar heights are expressed relative to a sphere with the same radius as recommended by the IAU and IAG (Archinal and others, 2011, 2018). The only other reference surface currently in use is a sphere with a radius of 1,738,000 m for gravity field modeling (Hirt and Featherston, 2012; Williams and others, 2014); recent gravity models like the Gravity Recovery and Interior Laboratory (GRAIL) mission provide simple conversions to scale data to a lunar surface sphere with a radius of 1,737,400 m (Williams and others, 2014). Since the Moon's surface closely approximates a sphere, ellipsoidal surfaces have not been as widely adopted as a lunar surface and are therefore not discussed in this document.

Lunar Positional Format

Once the LRS is defined, a positional format can be chosen. A variety of lunar coordinate systems have been in use throughout history, which are mostly based in a latitude and longitude format. As discussed above, the ME reference frame from the JPL DE (Folkner and others, 2008) is a widely adopted reference frame across the scientific community, and their products use latitude and longitude to describe positions on the surface. There are many ways to report this information, but the most commonly used systems are planetocentric, planetographic, and planetodetic latitude and longitude formats. For spherical celestial bodies like the Moon, planetocentric and planetographic latitudes can be used interchangeably. Longitudes for a planetocentric system can span −180° to 180° westward and eastward from the prime meridian or 0° to 360° east from the prime meridian. The LRO mission, NASA, and international space agencies generally have adopted the use of 0° to 360° east longitudes for the Moon (LRO mission and LGCWG, 2008). We note that planetocentric 0° to 360° east longitudes can easily be converted to planetocentric −180° to 180° longitudes and vice versa. Planetodetic coordinates are not typically used for the Moon.

Because of longitudinal convergence at the poles, crew members operating at or near the poles will encounter cardinal direction ambiguity, that is, if standing at the south pole, every direction is north. Additionally, latitude and longitude systems can be confusing in comprehending distances traveled because differences in longitude are functions of latitude and because of non-conformal spacing between denominations everywhere except the equator. For instance, at lat 84° S., 1 minute is equal to a distance of 505 m north or south; however, it is equal to a distance of 528 m east or west. This necessitates the use of a standardized grid system to maintain distance and orientation.

Planetary Map Projections

The IAU WG recommendations do not extend to other standards essential for digital mapping, such as map projections (Davies and others, 1980; Archinal and others, 2011, 2018). The IAU WG has provided recommendations on planetographic and planetocentric coordinate systems (Davies and others, 1980; Archinal and others, 2011, 2018). Many planetary map projections, including transverse Mercator and polar stereographic, are available to the scientific community under geodetic well-known text (WKT) using IAU-based projection codes (for example, IAU_2015_*) in most Geographic Information System (GIS) software packages (Hare and Malapert, 2021). IAU-based projections were created to support simple Open Geospatial Consortium web-map services and do not intend to address specific Artemis mission needs (Hare and Malapert, 2021; McClernan, 2023). Aside from the work of Hare and Malapert (2021), the only lunar mapping standardization document related to map projections is an Apollo-era transverse Mercator system (Kinsler, 1975) that has not been widely adopted.

With renewed interest in lunar exploration, standardization with modernized map projection design is needed to support the Artemis missions. The limitation for these IAU-based codes includes an unmodeled central scale factor set to 1 and the lack of a false easting and false northing (Hare and Malapert, 2021). Without defining these variables, the map distortion is high, and the projection is unable to support grid systems like LGRS as they use modular arithmetic (NGA, 2014b). The LTM and LPS systems as designed in this document provide projections needed to support a navigational grid system.

General Discussion on Navigation Standard

Parameters and Reference System

For this work, we have utilized the JPL lunar DE 421 ME reference frame as the basis of the lunar navigational system. As use of this frame is still recommended, we have used this existing positional standard (LRO mission and LGCWG, 2008; Archinal and others, 2018).

For this project, the LTM and LPS systems and the LGRS all use the current ME reference frame as the current PDS standard for lunar data archiving has a chosen reference frame for this lunar navigation standard (JPL, 2009); additionally, it is the most widely used and agreed upon reference frame for lunar cartographic data (LRO mission and LGCWG, 2008; Archinal and others, 2011, 2018). See Song and others (2021) for detailed summary of the current lunar reference frames.

In keeping with the recommendations of the LRO mission and LGCWG (2008), and Archinal and others (2011, 2018), this work utilizes a lunar sphere with a radius of 1,737,400 m. Many existing equations available for geodesy expect a reference ellipsoid. If an ellipsoid is used, we still utilize the lunar sphere because the reference surface expresses no ellipsoidal flattening ($f=0$) and therefore no eccentricity ($e=0$). If these values are zeroed, an ellipsoid is simplified to a sphere. The PCRS equations support ellipsoidal coordinates if a lunar reference ellipsoid is used in the future (Snyder, 1987; Karney, 2011).

The LTM and LPS systems, the LGRS, and the ACC condensed grid format are explained in later sections. The base unit is meters, and all computer-processed values are treated with double floating-point precision.

Planetocentric Coordinates

For this project, the positional format of the Moon utilizes planetocentric coordinates over the latitudinal interval of 90° to −90° and the longitudinal interval of −180° to 180° where the 0° prime meridian is the ME *x*-axis facing the Earth (fig. 2)

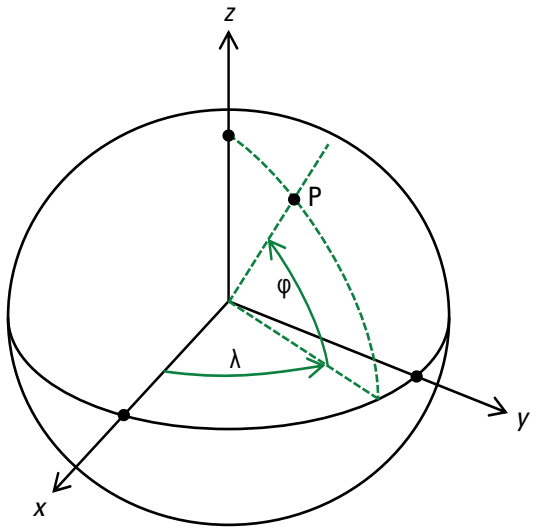


Figure 2. Plot showing the planetocentric coordinate layout: longitude, λ, increases positively eastward (to 180°) and negatively westward (to –180°) from the prime meridian; latitude, φ, increases positively northward (to 90°) and negatively southward (to –90°) from the equator. From Jet Propulsion Laboratory (2023).

Transverse Mercator and Lunar Transverse Mercator (LTM) System

This section outlines the specifications for the LTM coordinate system for the lunar surface (covering non-polar regions). The LTM system is similar in design to the UTM system (NGA, 2014b). As UTM is a globally used terrestrial navigation system and a current U.S. navigational standard (NGA, 2014b), a modernized version of LTM can be utilized for lunar navigation.

UTM, like LTM, is based on the transverse Mercator projection (Snyder, 1987; Karney, 2011, NGA, 2014b). Unlike the standard Mercator projection, which has its cylinder oriented along the equator, the transverse Mercator projection places the map surface on that of a horizontal cylinder, with an axis perpendicular to the equator. The transverse Mercator projection is a conformal map projection (Snyder, 1987; Karney, 2011). Conformal map projections maintain angular orientation while minimizing distortion in distance. Transverse Mercator is a well-modeled projection widely used for navigation and is used for all LTM zones (Snyder, 1987; Karney, 2011, NGA, 2014b).

Previous Defense Mapping Agency Apollo-Era LTM System

A historic LTM system was originally published by the DMA (NASA, 1973; Kinsler, 1975); however, the DMA topographic orthophoto chart series does not explicitly mention that the LTM system was used in maps during the Apollo exploration era. The LTM system described by Kinser (1975) has not been widely adopted or implemented. We recommend that the LTM system described in this document supersede and replace the system published in Kinsler (1975).

Definition of Transverse Mercator and LTM

To complete the conversion from planetocentric coordinates to the transverse Mercator projection, a single set of equations can be used. We convert planetocentric coordinates using Karney-Krüger transverse Mercator projection equations provided by Karney (2011). Karney (2011) based his work on Krüger (1912) for a modernized and reviewed set of transverse Mercator equations named Karney-Krüger equations. These equations (1) have high reprojection accuracy as great as 5 nm of coordinate conversion error on Earth; and (2) are able to map projection zones wider than 6° (Karney, 2011). McClernan and others (2023) recommend the use of these equations for correct positioning; however, other methods may be capable of supporting 8° LTM zones.

The basic definition of the LTM system uses a small number of agreed-upon concepts and set parameters to define projection areas which govern the Karney-Krüger equations. We discuss the parameters used in the transverse Mercator projection and LTM system and present the Karney-Krüger equations in this section.

Transverse Mercator and LTM Coordinate Structure

A conversion from latitude, φ, and longitude, λ, to LTM coordinates, produces the rectangular coordinates of zone, *Z*; hemisphere, *H*; easting, *E*; and northing, *N*. The grid scale factor and grid convergence can also be computed. We note that these LTM coordinate values are similar in structure to non-NATO-style UTM coordinates and do not include a latitudinal band.

In field communication of position, numerical values can be reported as a coordinate pair without the zone or hemisphere included; however, if the zone is included then it is reported before the hemisphere (that is, 12N). Easting is always reported before northing, to avoid confusion between these numerical values. Alternatively, when reporting an LTM coordinate, "easting" and "northing" can optionally be stated after their corresponding coordinate value.

When converting from LTM back to latitude and longitude, the zone and hemisphere need to be provided to properly complete the inverse conversion.

Transverse Mercator and LTM North-South Range

LTM is designed for optimal use between latitudes −80° and 80°. At latitudes south of −80° and north of 80°, LPS should be utilized. Additionally, a 2° buffer area, referred to as the "LTM extended range" can be utilized for areas that extend into a polar region and LPS zone. The LTM extended region can be utilized if the latitude is not south of −82° or north of 82°. The extended range is not considered the primary use area of the LTM system; however, both LTM and LPS coordinates are valid between latitudes −82° and −80° and between latitudes 80° and 82°. The LPS system is recommended for all polar regions.

Transverse Mercator and LTM East-West Range

The transverse Mercator projection has a limited usable width, often measured in degrees and referred to as "zonal width." There are two limiting factors in the usable width of a transverse Mercator projection: (1) the mathematics behind the map projection, and (2) scale error great enough to distort the ground surface such that the map is no longer practically usable. Each limiting factor is briefly discussed here:

1. The usable zone width for a transverse Mercator projection varies depending on the equation sets utilized. In terrestrial applications, the Thomas or Redfearn Series are utilized and limit transverse Mercator zone widths to 4° to 6° (Lee, 1945; Redfearn, 1948; Thomas, 1952). Krüger's (1912) methods can support a transverse Mercator zone width as wide as 20° with nanometer reprojection accuracy (Krüger, 1912; Karney, 2011). It is recommended that LTM utilize Karney-Krüger equations to accommodate larger zone widths and a higher coordinate conversion accuracy (see Karney, 2011).

2. Despite the increased usable width in using Karney-Krüger equations; there is a practicality limit. At a certain range, the scale error is considered too high, and the ground surface on the map is so heavily exaggerated that it is no longer useable despite the projection being mathematically correct and conformal (Snyder 1987). A simple accommodation for this high distortion is to use a different map projection of the same type, referred to as "switching zones," so that a less distorted projection can be utilized.

The LTM system is designed to maximize zone coverage while maintaining a reasonable scale error on the lunar reference sphere. As such, LTM zones are limited to a width of 8°. Of note are the surface distances across the projection area at different latitudes. The haversine formula is used to calculate lunar surface arc distances between two points:

$$\theta = \sin^2(\Delta\varphi/2) + \cos(\varphi_1)\cos(\varphi_2)\sin^2(\Delta\lambda/2), \tag{10}$$

$$c = 2 \times \tan 2^{-1}\left(\sqrt{\theta}, \sqrt{1-\theta}\right), \tag{11}$$

and

$$d = c \times a, \tag{12}$$

where

d	is the surface distance in meters,
θ	is the square of the half-chord length,
c	is the central angle,
$\Delta\varphi$	is the difference in latitude between the two points,
$\Delta\lambda$	is the difference in longitude between the two points,
φ_1 and φ_2	are latitudes of the two points, and
a	is the radius of the Moon in meters.

All angular measurements are in radians. The surface distances across an 8° zone at three key latitudes are as follows:

- at the lunar equator, 0°, the zone spans a surface distance of 243 km,
- at latitude ±80° the zone spans a surface distance of 42 km, and
- at latitude ±82° the zone spans a surface distance of 33 km.

If the projection border were placed further towards the lunar poles, the transverse Mercator projection would become impractical owing to the small area and high distortions imposed on the map projection area. Using this projection, the scale error just exceeds the ±1:1,000 recommendation in equatorial regions. The areas that exceed this recommendation are located at the fringes of the 8° zones near the lunar equator.

Transverse Mercator and LTM Zones

Transverse Mercator zones subdivide the Moon by longitude into small slices where the distortion is considered acceptable. Using 8°-wide transverse Mercator zones for the LTM system subdivides the lunar surface into 45 equally spaced zones. Subdividing further into north and south lunar hemispheres, the LTM portion of the LGRS system has 90 zones (fig. 3).

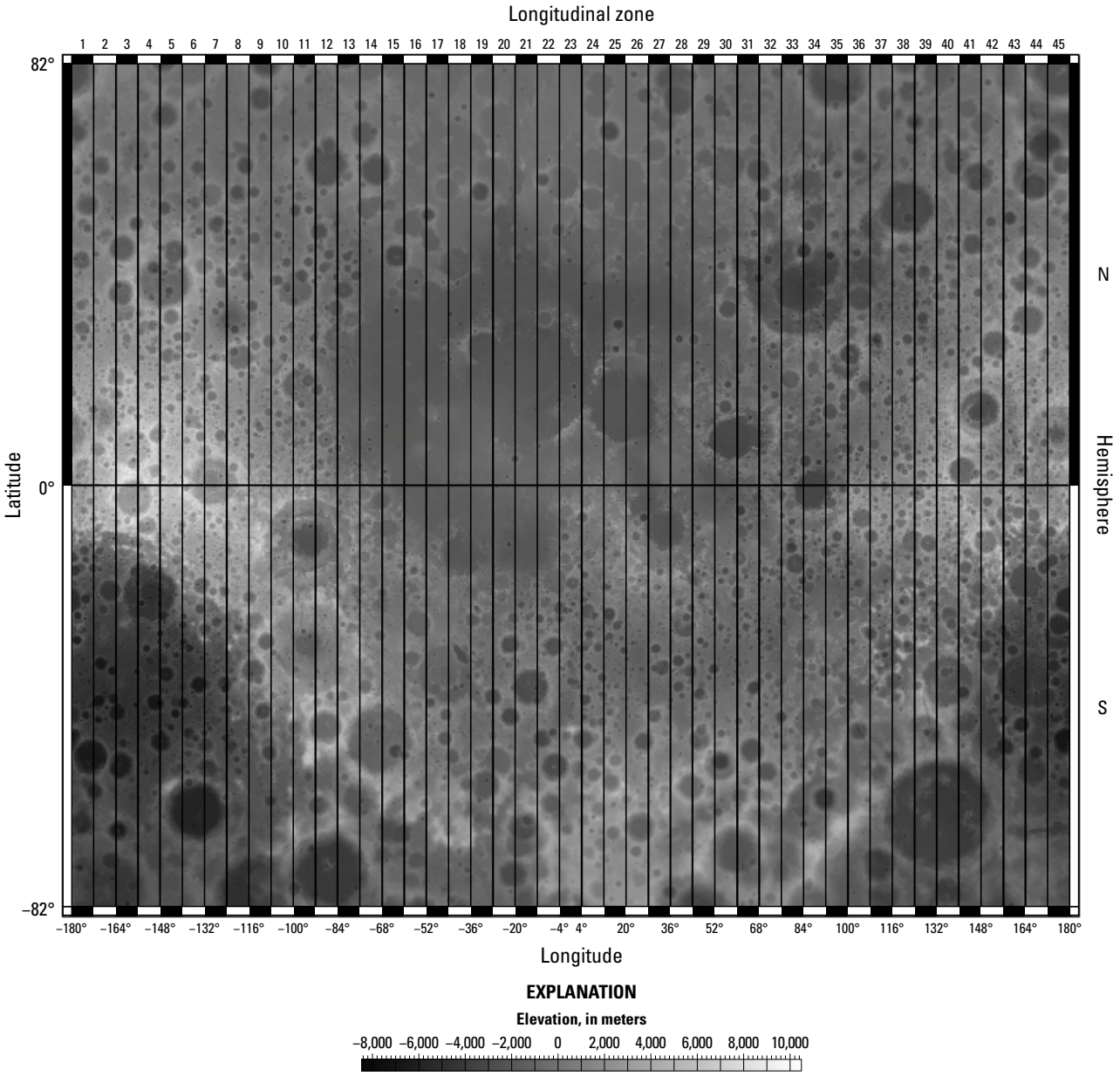


Figure 3. Map showing the Lunar Transverse Mercator (LTM) 8°-zone and hemisphere layout in a Mercator projection. Elevation from a Lunar Orbiter Laser Altimeter (LOLA) 118-meter (m)-resolution digital elevation model (LOLA Science Team, 2021). See tables 2 and 3 for east–west and north–south extents of the LTM zones.

In comparison, UTM uses 6°-wide transverse Mercator zones, subdividing earth into 120 zones between the north and south hemispheres on Earth (NGA, 2014b). Using 6°-wide or narrower zones for the LTM system was not prioritized as this would increase the overall number of zones. As the radius and reference surface of the Moon are much smaller than those of the Earth, smaller zones would greatly increase the number of potential zone changes over a given distance owing to the smaller zone surface coverage. This is especially notable close to the poles where the surface distance across a zone is much smaller.

The zone number, Z, can be determined with floor division as follows:

$$Z = \left\lfloor \frac{\lambda + 180°}{2W} \right\rfloor + 1 , \tag{13}$$

where

$2W$ is twice the zonal width from the prime meridian, and

λ is the planetocentric longitude from −180° to 180° as an integer value.

The hemisphere, H, can be determined to be northern (“N”) or southern (“S”) if

$$\varphi < 0°, |H = \text{"S"}| \tag{14}$$

or

$$\varphi \geq 0°, |H = \text{"N"}| . \tag{15}$$

LTM Zone Extents

The 90 zones in the LTM system are referenced by zone number, Z, and a letter designating the hemisphere, H, which are combined into a string, ZH. Each zone is 8° wide or 4° east and west from its central meridian. Table 2 defines the east–west extents of each zone by zone number. We note table 2 only has 45 zone number entries—these zones are subdivided between the north and south hemispheres by appending either a letter N or S to the zone number for a total of 90 zones. Table 3 defines the north–south extent of each hemisphere.

Table 2. East–west extent of each zone, by zone number, in the Lunar Transverse Mercator (LTM) system.

[Boundaries are longitudes in degrees west and east]

LTM zone	Western boundary	Eastern boundary	LTM zone	Western boundary	Eastern boundary
1	−180°	−172°	23	−4°	4°
2	−172°	−164°	26	20°	28°
3	−164°	−156°	27	28°	36°
4	−156°	−148°	28	36°	44°
5	−148°	−140°	29	44°	52°
6	−140°	−132°	30	52°	60°
7	−132°	−124°	31	60°	68°
8	−124°	−116°	32	68°	76°
9	−116°	−108°	33	76°	84°
10	−108°	−100°	34	84°	92°
11	−100°	−92°	35	92°	100°
12	−92°	−84°	36	100°	108°
13	−84°	−76°	37	108°	116°
14	−76°	−68°	38	116°	124°
15	−68°	−60°	39	124°	132°
16	−60°	−52°	40	132°	140°
17	−52°	−44°	41	140°	148°
18	−44°	−36°	42	148°	156°
19	−36°	−28°	43	156°	164°
20	−28°	−20°	44	164°	172°
21	−20°	−12°	45	172°	180°
22	−12°	−4°			

Table 3. North–south maximum extent of each hemisphere in the Lunar Transverse Mercator (LTM) system used to distinguish northern and southern zones.

LTM hemisphere	Northern Boundary	Southern Boundary
North, H="N"	≤ 82°	≥ 0°
South, H="S"	< 0°	≥ −82°

Transverse Mercator and LTM Projection Axis

Map projections define a one-to-one mathematical relationship between points located on the reference surface and their representation on a plane (Stem, 1990; Dennis, 2022). Transverse Mercator map projections produce a rectangular mapping system projected from an axis, the central meridian, across which the central scale factor is a constant, minimum value (Stem, 1990; Dennis, 2022).

Each LTM zone of the Moon's surface uses a different transverse Mercator projection with a unique projection axis located along the lunar equator. The projection axis for each projection is oriented north–south and is aligned with the central meridian of each zone. The origin of the map projection is centered on reference longitude λ_0 and reference latitude φ_0. The projection axis of each zone is defined using the following equations:

$$\lambda_0 = (Z-1) \times 2W - 180° + W, \tag{16}$$

and

$$\varphi_0 = 0°, \tag{17}$$

where φ_0 is always located along the lunar equator for both northern and southern LTM zones. Many software implementations define W as distance from the prime meridian; we do the same in LTM. To determine the correct 8° LTM zone with equation 16, W=4°.

Transverse Mercator and LTM Central Scale Factor

Map projection distortion is an inevitable outcome of representing a 3D curved surface on a 2D map (Stem, 1990; Dennis, 2022). Distortion cannot be eliminated from a map projection; however, its effect can be minimized and balanced across the map area. At the projection axis, the scale factor, k, is at its minimum and set via the central scale factor, k_0. Distortion and scale error generally increase from and with increasing arc-distances from the projection axis.

For the transverse Mercator projection, scale error increases with arc-distance from the zone's central or reference meridian (Snyder, 1987, NGA, 2014b). Given the recommendation for a map projection within a ±1:1,000 scale error, we set the central scale factor to 0.999 (exactly) using a double floating-point value for each LTM projection. This scale error allows a zone width of just less than 8°. Areas that exceed the 1:1,000 scale error recommendation are at the zone fringes near the equator.

Transverse Mercator and LTM False Origins

The LTM system applies false origins in its coordinate system. False origins are used to shift the coordinate value from the reference axis location to the lower left-hand corner of the projection area. This shift in coordinate values is completed so coordinates are always positive and never change the number of significant digits. False origins allow for more consistent communication and easier verbal communication of distances. We utilize a false easting (F_E) of 250 km (updated from McClernan and others, 2023), and a false northing (F_N) of 2,500 km, which were similarly used in the historic LTM system (Kinsler, 1975). Note that a false northing is only utilized for LTM zones in the southern hemisphere (H="S"). If a point is in the northern hemisphere, the equator represents 0 m on the LTM grid in place of a false northing.

LTM System Conversion Equations

This section presents the formulas for a transverse Mercator map projection as defined by Karney-Krüger equations and as implemented in the LTM system (McClernan and others, 2023). See Snyder (1987) and Karney (2011) for more discussion of transverse Mercator map projection properties and equations.

LTM Parameters and Formulization

The required parameters for the LTM system are listed in table 4 with their associated symbols, values, and units. These values are unique for the north and south hemispheres and differ slightly. Note that the Karney-Krüger equations were developed for an ellipsoid, thus some terms are not applicable to a sphere. These terms are zeroed for the LTM system given that the Moon's reference surface is a sphere.

Table 4. Lunar Transverse Mercator (LTM) system parameters.

Parameter	Symbol	Value	Units
Northern hemisphere zones			
Central scale factor	k_0	0.999	Unitless
LTM zone half width	W	4	Arc-degrees
Latitude of projection axis	φ_0	0	Arc-degrees
Longitude of projection axis	λ_0	Specific to each zone[1]	Arc-degrees
False easting	F_E	250,000	Meters
False northing	F_N	0[2]	Meters
Lunar radius	a	1,737,400	Meters
Ellipsoidal flattening	f	0	Unitless
Eccentricity	e	$\sqrt{f\times(2-f)}$	Unitless
Third flattening	n	$f/(2-f)$	Unitless
Southern hemisphere zones			
Central scale factor	k_0	0.999	Unitless
LTM zone half width	W	4	Arc-degrees
Latitude or projection origin	φ_0	0	Arc-degrees
Longitude of projection origin	λ_0	Specific to each zone[1]	Arc-degrees
False easting	F_E	250,000	Meters
False northing	F_N	2,500,000[2]	Meters
Lunar radius	a	1,737,400	Meters
Ellipsoidal flattening	f	0	Unitless
Eccentricity	e	$\sqrt{f\times(2-f)}$	Unitless
Third flattening	n	$f/(2-f)$	Unitless

[1]See equation 16 to calculate the projection axis longitude for a specific zone.

[2]A false northing is not applied in northern hemisphere LTM system zones.

Forward Conversion $\{\lambda,\varphi\}\rightarrow\{X,Y\}$

This section outlines the steps used to convert a point location from planetocentric latitude and longitude to the LTM grid. The steps are modified from Karney (2011). We refer to the forward conversion from latitude and longitude to LTM coordinates with the following general mapping statement:

$$LatLon(\lambda,\varphi)\overset{LTM}{\rightarrow}\{E,N,H,Z\}, \tag{18}$$

where the latitude, φ, and longitude, λ, are converted to the LTM rectangular coordinates of zone, Z; hemisphere, H; easting, E; and northing, N. The parameters of the map projection used in this calculation are a, f, k_0, λ_0, W, F_E, and F_N.

Step 1.—Determine if the point location is within the LTM range or extended LTM range:

$$LTM:\{-80°\le\varphi\le 80°\}, \tag{19}$$

or

$$LTM\_ext:\{-82°\le\varphi\le 82°\}. \tag{20}$$

Generally, the LPS system should be used for polar locations in the extended LTM range; the extended range needs to be specified if converting to the LTM grid.

Step 2.—Compute the ellipsoid constants e^2 and n:

$$e^2 = f\left(2 - f\right), \tag{21}$$

and

$$n = \frac{f}{2 - f}. \tag{22}$$

Note that the current reference surface of the Moon is a spheroid so $f = 0$, and therefore $e^2 = 0$, $n = 0$.

Step 3.—Determine the LTM zone and hemisphere. The LTM zone can be calculated using the following equation:

$$Z = \left\lfloor \frac{\left(\lambda + 180°\right)}{2W} \right\rfloor + 1, \tag{23}$$

where

$\lfloor \cdot \rfloor$ is the floor.

The hemisphere can be determined simply from

$$H = \begin{Bmatrix} \varphi \geq 0° & \text{"N"} \\ \varphi < 0° & \text{"S"} \end{Bmatrix}. \tag{24}$$

The reference meridian (if one has not already been determined) can be calculated using equation 16.

Step 4.—Convert latitude and longitude to radians and compute the difference between the reference meridian and the point location latitudes:

$$\omega = \lambda - \lambda_0. \tag{25}$$

Step 5.—Compute the rectifying radius of the spheroid, A, from the series truncated to a sixth order:

$$A = \frac{a}{1+n} \times \left(1 + \left(\frac{1}{4}\right) \times n^2 + \left(\frac{1}{64}\right) \times n^4 + \left(\frac{1}{256}\right) \times n^6\right). \tag{26}$$

Using double floating-point precision and sixth-order equations is recommended by Karney (2011) for optimal, high-accuracy positioning. Increasing to an eighth order improves accuracy; however, it refines the coordinate at nanometer scale and has little effect on the final position (Karney, 2011).

Step 6.—Determine the coefficients α_{2k} to a sixth order where subscript $k = \{1, 2, \ldots, 6\}$:

$$\alpha_2 = \left(\frac{1}{2}\right) n - \left(\frac{2}{3}\right) n^2 + \left(\frac{5}{16}\right) n^3 + \left(\frac{41}{180}\right) n^4 - \left(\frac{127}{288}\right) n^5 + \left(\frac{7891}{37800}\right) n^6, \tag{27}$$

$$\alpha_4 = \left(\frac{13}{48}\right) n^2 - \left(\frac{3}{5}\right) n^3 + \left(\frac{557}{1440}\right) n^4 + \left(\frac{281}{630}\right) n^5 - \left(\frac{1983433}{1935360}\right) n^6,$$

$$\alpha_6 = \left(\frac{61}{240}\right) n^3 - \left(\frac{103}{140}\right) n^4 + \left(\frac{15061}{26880}\right) n^5 + \left(\frac{167603}{181440}\right) n^6,$$

$$\alpha_8 = \left(\frac{49561}{161280}\right)n^4 - \left(\frac{179}{168}\right)n^5 + \left(\frac{6601661}{7257600}\right)n^6,$$

$$\alpha_{10} = \left(\frac{34729}{80640}\right)n^5 - \left(\frac{3418889}{1995840}\right)n^6,$$

$$\alpha_{12} = \left(\frac{212378941}{319334400}\right)n^6.$$

Step 7.—Compute conformal latitude, φ':

$$\tan\left(\varphi'\right) = \tan\left(\varphi\right)\sqrt{1+\sigma^2} - \sigma\sqrt{1+\tan\left(\varphi\right)}, \tag{28}$$

where

$$\sigma = \sinh\left(e \times \tanh^{-1}\left(\frac{e \times \tan\left(\varphi\right)}{\sqrt{1+\tan^2\left(\varphi\right)}}\right)\right),$$

and e is eccentricity.

Step 8.—Compute the Gauss-Schreiber ratios $\xi' = u/v$ and $\eta' = v/a$:

$$\xi' = \tan^{-1}\left(\tan\left(\frac{\varphi'}{\cos\left(\omega\right)}\right)\right), \tag{29}$$

and

$$\eta' = \sinh^{-1}\left(\frac{\sin\left(\omega\right)}{\sqrt{\tan^2\left(\varphi'\right)+\cos^2\left(\omega\right)}}\right). \tag{30}$$

Step 9.—Compute the transverse Mercator ratios $\xi = x/A$ and $\eta = y/A$ to a sixth order:

$$\xi = \xi' + \sum_{k=0}^{6}\alpha_{2k}\sin\left(2k\xi'\right)\cosh\left(2k\eta'\right), \tag{31}$$

and

$$\eta = \eta' + \sum_{k=0}^{6}\alpha_{2k}\cos\left(2k\xi'\right)\sinh\left(2k\eta'\right). \tag{32}$$

For computational efficiency, this summation is unrolled and completed without iteration.

Step 10.—Compute the transverse Mercator coordinates:

$$x = A\xi, \tag{33}$$

and

$$y = A\eta. \tag{34}$$

Step 11.—Apply false origins and a central scale factor to determine the final easting (E) and northing (N):

$$E = k_0 x + F_E, \tag{35}$$

and

$$N = k_0 y + F_N. \tag{36}$$

The forward conversion of λ and φ to transverse Mercator positions of Z, H, E, and N is complete.

The following steps describe how to compute the grid scale factor, k, and the grid convergence, γ.

Step 12.—Compute the factors p and q to a sixth order:

$$p = 1 + \sum_{k=0}^{6} 2k\alpha_{2k} \cos\left(2k\xi'\right)\cosh\left(2k\eta'\right), \tag{37}$$

and

$$q = \sum_{k=0}^{6} 2k\alpha_{2k} \sin\left(2k\xi'\right)\sinh\left(2k\eta'\right). \tag{38}$$

See Karney (2011) for a description of these terms.

Step 13.—Compute the grid scale factor, k:

$$k = k_0\left(\frac{A}{a}\right)\sqrt{p^2 + q^2}\left(\frac{\sqrt{1+\tan^2(\varphi)}\sqrt{1-e^2\sin^2(\varphi)}}{\sqrt{\tan^2(\varphi') + \cos^2(\omega)}}\right). \tag{39}$$

Step 14.—Compute the grid scale convergence, γ:

$$\gamma = \tan^{-1}\left(\left|\frac{q}{p}\right|\right) + \tan\left(\frac{\left|\tan(\varphi')\tan(\omega)\right|}{\sqrt{1+\tan^2(\varphi')}}\right). \tag{40}$$

Inverse Conversion $\{X, Y\} \rightarrow \{\lambda, \varphi\}$

This section outlines the steps used to convert a point location from the LTM grid to planetocentric latitude and longitude. We refer to the inverse conversion from LTM coordinates to latitude and longitude with the following general mapping statement:

$$LTM : (E, N, H, Z) \overset{LatLon:}{\rightarrow} \varphi, \lambda. \tag{41}$$

The parameters used during the forward conversion, $a, f, k_0, \lambda_0, W, F_E$, and F_N, are used during the inverse conversions.

The steps are modified from Karney (2011).

Step 1.—Calculate the ellipsoidal constants e^2 and n using equations 21 and 22. Note that the current reference surface of the Moon is a spheroid, and therefore $e^2 = 0$ and $n = 0$.

Step 2.—Compute the reference spheroid rectifying radius, A, to a sixth order as recommended by Karney (2011):

$$A = \frac{a}{1+n} \times \left(1 + \left(\frac{1}{4}\right) \times n^2 + \left(\frac{1}{64}\right) \times n^4 + \left(\frac{1}{256}\right) \times n^6\right). \tag{42}$$

Step 3.—Compute transverse Mercator ratios ξ and η by removing false origins, descaling the central scale factor, and dividing by the radius:

$$\eta = \frac{E - F_E}{k_0 A}, \tag{43}$$

and

$$\xi = \frac{N - F_N}{k_0 A}. \tag{44}$$

H is used to determine the false northing, F_N, value from table 4.

Step 4.—Compute the coefficient β_{2k} to a sixth order where subscript $k = \{1, 2, \ldots, 6\}$:

$$\beta_2 = \left(\frac{1}{2}\right)n - \left(\frac{2}{3}\right)n^2 + \left(\frac{37}{96}\right)n^3 - \left(\frac{1}{360}\right)n^4 - \left(\frac{81}{512}\right)n^5 + \left(\frac{96199}{604800}\right)n^6, \tag{45}$$

$$\beta_4 = \left(\frac{1}{48}\right)n^2 + \left(\frac{1}{15}\right)n^3 - \left(\frac{437}{1440}\right)n^4 + \left(\frac{46}{105}\right)n^5 - \left(\frac{1118711}{3870720}\right)n^6,$$

$$\beta_6 = \left(\frac{17}{480}\right)n^3 - \left(\frac{37}{840}\right)n^4 - \left(\frac{209}{4480}\right)n^5 + \left(\frac{5569}{90720}\right)n^6,$$

$$\beta_8 = \left(\frac{4397}{161280}\right)n^4 - \left(\frac{11}{504}\right)n^5 - \left(\frac{830251}{7257600}\right)n^6,$$

$$\beta_{10} = \left(\frac{4583}{161280}\right)n^5 - \left(\frac{108847}{3991680}\right)n^6,$$

$$\beta_{12} = \left(\frac{20648693}{638668800}\right)n^6.$$

Step 5.—Compute the Gauss Schreiber ratios $\xi' = u/a$ and $\eta' = v/a$:

$$\xi' = \xi - \sum_{k=0}^{6} \beta_{2k} \sin\left(2k\xi\right)\cosh\left(2k\eta\right), \tag{46}$$

and

$$\eta' = \eta - \sum_{k=0}^{6} \beta_{2k} \cos\left(2k\xi\right)\sinh\left(2k\eta\right). \tag{47}$$

Step 6.—Compute $t' = \tan\left(\varphi'\right)$:

$$t' = \tan\left(\varphi'\right) = \frac{\sin\left(\xi'\right)}{\sqrt{\sinh^2\left(\eta'\right) + \cos^2\left(\xi'\right)}}, \tag{48}$$

where

φ' is the conformal latitude.

Step 7.—Calculate the latitude. Latitude can be determined after computing $t = \tan\left(\varphi\right)$ and $t' = \tan\left(\varphi'\right)$ using modified versions of equations 28 and 48:

$$t' = t\sqrt{1 + \sigma^2} - \sigma\sqrt{1 + t^2}, \tag{49}$$

and

$$\sigma = \sinh\left(e \times \tanh^{-1}\left(\frac{e \times t}{\sqrt{1 + t^2}}\right)\right). \tag{50}$$

Determining *t* from this step requires iteration to apply the previous equations. Equations 51–53 are from Karney (2011); our implementation is written in equations 54–56. Evaluate *t* using the Newton-Raphson method for real roots in linearized form, given $f(t)=0$:

$$t_{i+1} = t_n - \frac{f(t_i)}{f'(t_i)}. \tag{51}$$

t_i denotes the ith iterative value and $f(t)$ is calculated by

$$f(t) = t\sqrt{1+\sigma^2} - \sigma\sqrt{1+t^2}. \tag{52}$$

The derivative value of $f(t)$, $f(t)' = \frac{d}{dt}\left(f(t)\right)$ calculated by

$$f'(t) = \left(\sqrt{1+\sigma^2}\sqrt{1+t^2} - \sigma t\right)\frac{\left(1+e^2\right)\sqrt{1+t^2}}{1+\left(1-e^2\right)t^2}. \tag{53}$$

An initial estimate of t_1 is taken from $t_1 = t' = \tan(\varphi')$, a precalculated fixed value denoted $f(t_1)$ and $f'(t_1)$.

Our iteration varies slightly from the above equations provided by Karney (2011). First, we assign starting values prior to iteration with the following equations:

$$dt_i = 1, \tag{54}$$

and

$$t_i = t' = t\sqrt{1+\sigma^2} - \sigma\sqrt{1+t^2}. \tag{55}$$

Iterate to solve for *t*:

$$t_i = \begin{cases} \sigma_i = sinh\left(e \times \tanh^{-1}\left(\dfrac{e \times t_i}{\sqrt{\left(1+t_i^2\right)}}\right)\right) \\ t_i' = t_i\sqrt{1+\sigma_i^2} - \sigma_i\sqrt{1+t_i^2} \qquad \text{if } dt_i < 10^{-13}. \\ dt_i = \dfrac{t'-t_i'}{\sqrt{1+t_i'^2}} \dfrac{1+\left(1-e^2\right)t_i^2}{\left(1-e^2\right)\sqrt{1+t_i^2}} \\ t_i += dt_i \end{cases} \tag{56}$$

Iterate until the adjustment value, dt_i, is smaller than 10^{-13}. This iteration converges very quickly and often this calculation is only iterated two or three times to complete. The round-off value is widely utilized for Newton-Raphson iteration (Karney, 2011). Note that $10^{-13°}$ of lunar latitude is approximately 3×10^{-9} m on the lunar surface, or 3 nm, an acceptable positional error tolerance when converting to latitude and longitude. As an added safety measure, we terminate iteration when *i* is greater than 50 if the solution has not converged. In any implemented software, proper checks and processing fail-safes should be taken so that the inverse conversion of coordinates is always able to converge. We have placed similar safeguards in the example code provided with this document and an error will be reported if the solution does not converge.

Once t is determined calculate latitude:

$$\varphi = \tan^{-1}(t). \tag{57}$$

Step 8.—Compute the difference in longitude from the prime meridian:

$$\omega = \tan 2^{-1}\left(\sinh(\eta'), \cos(\xi')\right), \tag{58}$$

and

$$\lambda = \lambda_0 + \omega. \tag{59}$$

The inverse conversion of coordinates from Z, H, E, and N to λ, φ is complete. At this point the values are in radians; a final conversion from radian to degrees can be applied to determine the final planetocentric latitude and longitude values.

The following steps describe how to compute the grid scale factor, *k*, and the grid convergence, γ.

Step 9.—Compute the factors *p* and *q* to a sixth order:

$$p = 1 + \sum_{k=0}^{6} 2k\alpha_{2k} \cos\left(2k\xi'\right)\cosh\left(2k\eta'\right), \tag{60}$$

and

$$q = \sum_{k=0}^{6} 2k\alpha_{2k} \sin\left(2k\xi'\right)\sinh\left(2k\eta'\right). \tag{61}$$

Step 10.—Compute the grid scale factor, *k*:

$$k = k_0\left(\frac{A}{a}\right)\sqrt{p^2 + q^2}\left(\frac{\sqrt{1+\tan^2(\varphi)}\sqrt{1-e^2\sin^2(\varphi)}}{\sqrt{\tan^2(\varphi') + \cos^2(\omega)}}\right). \tag{62}$$

Step 11.—Compute the grid scale convergence, γ:

$$\gamma = \tan^{-1}\left(\left|\frac{q}{p}\right|\right) + \tan\left(\frac{\left|\tan(\varphi')\tan(\omega)\right|}{\sqrt{1+\tan^2(\varphi')}}\right). \tag{63}$$

LTM System Sample Conversion and Grid Generation Program

Preliminary software to complete the forward and inverse LTM system coordinate conversions is provided in appendix 1. Additional equations to create shapefiles of the LTM system are provided in appendix 2.

LTM Map Projection and Sample WKT

Here we include sample WKT for LTM zone 23 in the north and south hemispheres (fig. 3). These projections are not included in Esri's ArcGIS, QGIS, the European Petroleum Survey Group (EPSG) registry, or the IAU projection library (Hare and Malapert, 2021); however, this text can be copied and pasted into most GIS software projection libraries to reproduce the LTM projections.

WKT Sample for LTM Zone 23N

```
PROJCRS["Moon (2015) - Sphere / Ocentric / Tranverse Mercator",
  BASEGEOGCRS["Moon (2015) - Sphere / Ocentric",
    DATUM["Moon (2015) - Sphere",
      ELLIPSOID["Moon (2015) - Sphere",1737400,0,
        LENGTHUNIT["metre",1]]],
    PRIMEM["Reference Meridian",0,
      ANGLEUNIT["degree",0.0174532925199433]],
    ID["IAU",30100,2015]],
  CONVERSION["transverse Mercator",
    METHOD["transverse Mercator",
      ID["EPSG",9807]],
    PARAMETER["Latitude of natural origin",0,
      ANGLEUNIT["degree",0.0174532925199433],
      ID["EPSG",8801]],
    PARAMETER["Longitude of natural origin",0,
      ANGLEUNIT["degree",0.0174532925199433],
      ID["EPSG",8802]],
    PARAMETER["Scale factor at natural origin",.999,
      SCALEUNIT["unity",1],
```

```
        ID[“EPSG”,8805]],
      PARAMETER[“False easting”,250000,
        LENGTHUNIT[“metre”,1],
        ID[“EPSG”,8806]],
      PARAMETER[“False northing”,0,
        LENGTHUNIT[“metre”,1],
        ID[“EPSG”,8807]]],
    CS[Cartesian,2],
      AXIS[“(E)”,east,
        ORDER[1],
        LENGTHUNIT[“metre”,1]],
      AXIS[“(N)”,north,
        ORDER[2],
        LENGTHUNIT[“metre”,1]],
    USAGE[
      SCOPE[“Lunar Transverse Mercator Zone LTM_23N.”],
      BBOX[82.,-4,0.,4]]]
```

WKT Sample for LTM Zone 23S

```
LTM Zone 23S WKT Sample
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_23S.”],
    BBOX[0.,-4,-82.,4]]]
```

WKT for All LTM Zones

See appendix 3 for WKT for all 90 LTM map projections.

LTM Surface Distortion Analysis

The tolerance for map projection scale error specified by recommendation 1 from NASA's Flight Operations Directorate to the AGDT (McClernan and others, 2023) is ±1:1,000 or 1 m/km. This recommendation translates to a grid scale factor between 0.999 and 1.001. This target range is achievable for scale errors calculated on the lunar reference sphere. Traditionally, map scale error is calculated on the Earth's reference ellipsoid and we have completed a similar calculation for the Moon. Additionally, we have calculated the combined factor for each LTM zone. Accounting for the combined factor from topography and the map projection is important when considering distortion and the associated changes between grid and ground distances. As such, calculating the combined factor is greatly important; especially in the lunar south pole where topographic relief is very high. For more information on distortion in lunar polar regions see the "LPS Surface Distortion Analysis" section.

To analyze the scale error and distortion imposed by the LTM system we calculated the grid scale factor, k, using equation 62 and then used this to calculate the combined factor, C_F, for each LTM zone. As the figures below show, this target scale error is achieved for most of the LTM region with the exception of areas within ±25° of latitude of the lunar equator. Note that when interpreting the maps, any data outside the previously mentioned latitude range meets the ±1:1,000 scale error recommendation. Between latitudes −25° and 25°, the scale error exceeds +1:1,000 ($k > 1.001$) at the boundaries between LTM zones along the equator; the maximum scale factor is +1:1.00144 (fig. 4). A higher scale factor near the equator was considered acceptable to complete the LTM system by NASA and the AGDT (Flight Operations Directorate, NASA, oral commun., 2023). This slight increase in error tolerance allows for wider zones that cover larger surface distances in the lunar polar regions. Strictly restricting the scale error to ±1:1,000 would require narrower zones and greatly increase the overall number of zones, making LTM implementation near the poles less ideal. Using 8° zones with a central scale factor of 0.999 covers the most area per zone with as few map projections as possible.

We note that scale error for the UTM system used on Earth exceeds ±4:10,000 at the equator. UTM has a central scale factor of 0.9996 at its central meridians; the grid scale factor increases to a maximum value of 1.0010 at the Earth's equator along the UTM zone borders. This value was calculated using equation 62 with WGS84 parameters (NGA, 2014b). LTM exceeds its scale error range in a similar manner to UTM.

For a more rigorous determination of the impact of the LTM system and its errors, we have extended the calculation to determine distortion at the topographic surface and have computed the height factor (eq. 2) and used this to compute the combined factor (eq. 1), which accounts for the distortion of the topography and the scale factor of the transverse Mercator map projection (eq. 62).

The height factor accounts for the variations in elevation across the map projection and compensates for changes in terrain. We used a lunar sphere radius of 1,737,400 m and a Lunar Orbiter Laser Altimeter (LOLA) 118-m-resolution gridded digital elevation model (DEM) (LOLA Science Team, 2021) to calculate the height factor using equation 2 (fig. 5). Note that the height factor is independent from the map projection. Negative topography expresses a height factor greater than 1, indicating that grid distances are longer than ground distances. Positive topography expresses a height factor less than 1, indicating that grid distances are shorter than ground distances. As shown in figure 5, the height factor across the equatorial regions of the Moon ranges from 0.994 to 1.005. Discussion of distortion in relation to specific geographic regions is beyond the scope of this document.

The combined factor (the product of grid scale factor and height factor, eq. 1), accounts for both the vertical and horizontal scale variations across the map area. Similarly to the height factor, a combined factor greater than 1 indicates that grid distances are longer than ground distances, and a combined factor less than 1 indicates that grid distances are shorter than ground distances.

The combined factor ranges from 0.9932 to 1.0050 across all LTM zones (fig. 6). As shown in figure 6, the combined factor from the LTM system is greatest at boundaries between LTM zones near the equator; at these locations high scale error values from the LTM projection combines with topographic distortion to produce a greater combined factor. However, at polar latitudes the topography is the dominant contributor to the combined factor. The effect to measuring distance on a map area is extremely large. A combined factor range of 0.9932 to 1.0050 corresponds to a linear distortion of −6.8 m/km and +5.0 m/km. Discussion of distortion in relation to specific geographic regions is beyond the scope of this document.

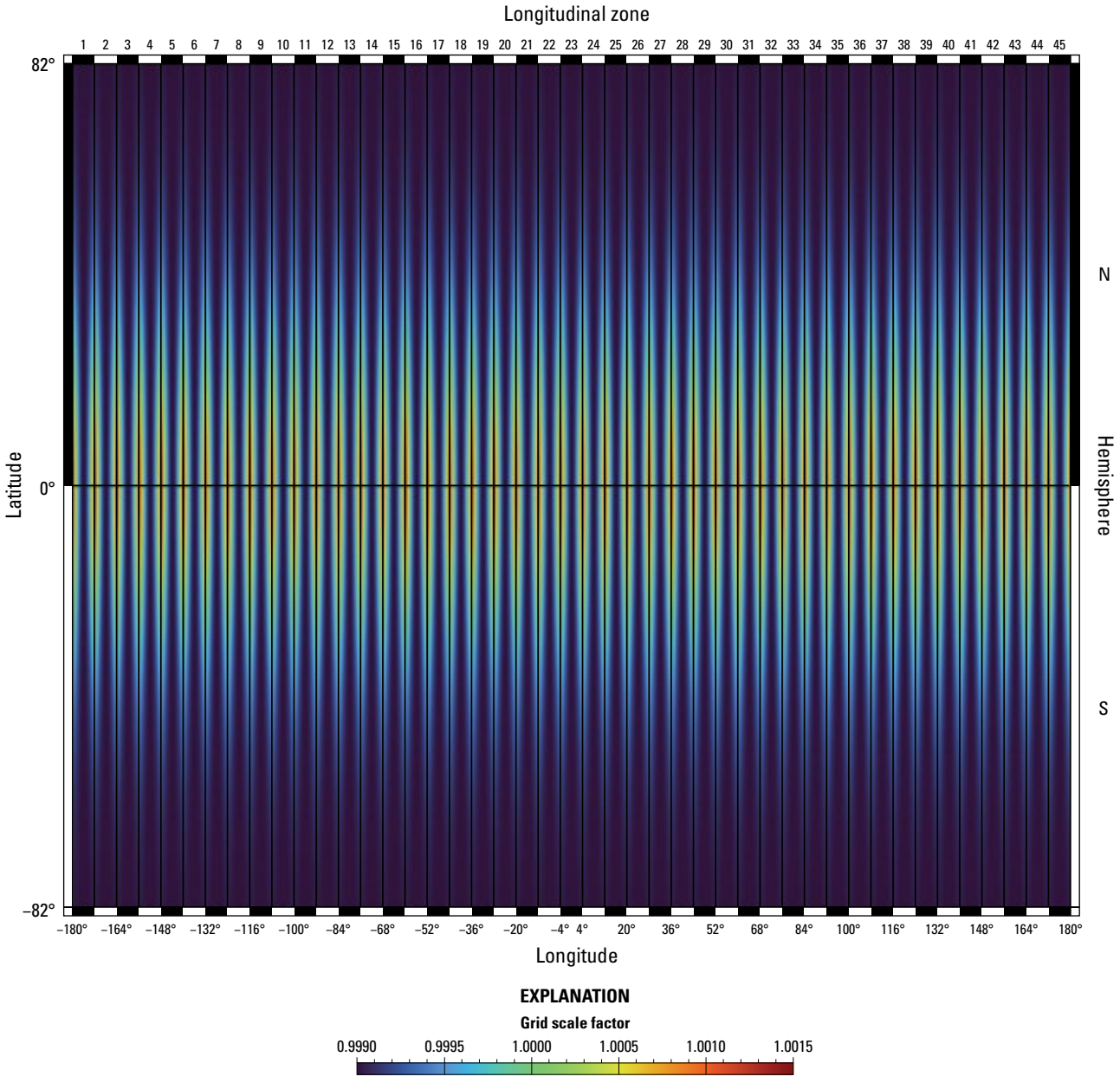


Figure 4. Map showing grid scale factor calculated for the transverse Mercator projections of the 90 zones in the Lunar Transverse Mercator (LTM) system. See tables 2 and 3 for east–west and north–south extents of the LTM zones.

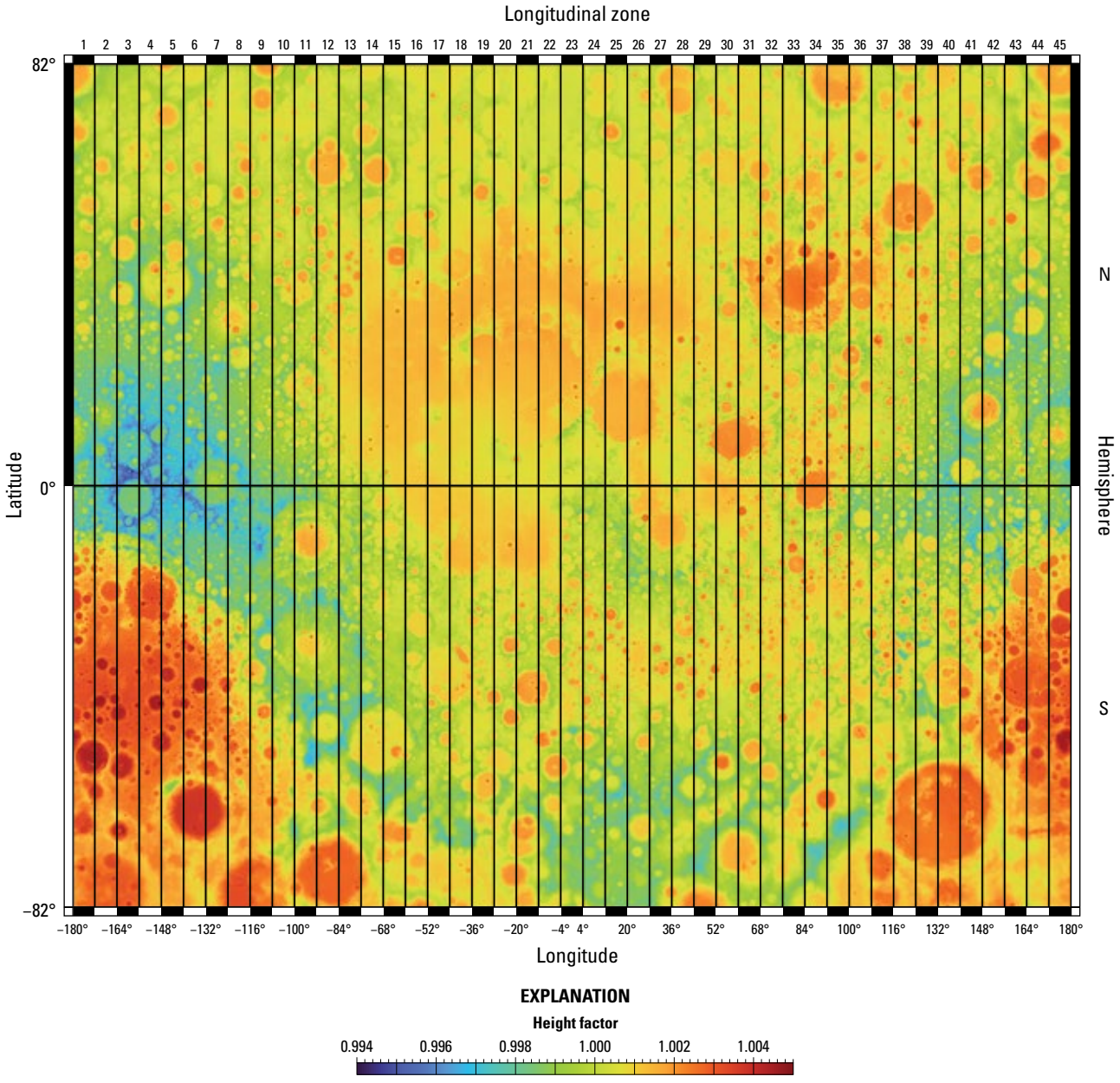


Figure 5. Map showing height factor calculated for the region of the Lunar Transverse Mercator (LTM) system. Calculated from lunar elevation data from a Lunar Orbiter Laser Altimeter (LOLA) 118-meter-resolution digital elevation model (LOLA Science Team, 2021). See tables 2 and 3 for east–west and north–south extents of the LTM zones.

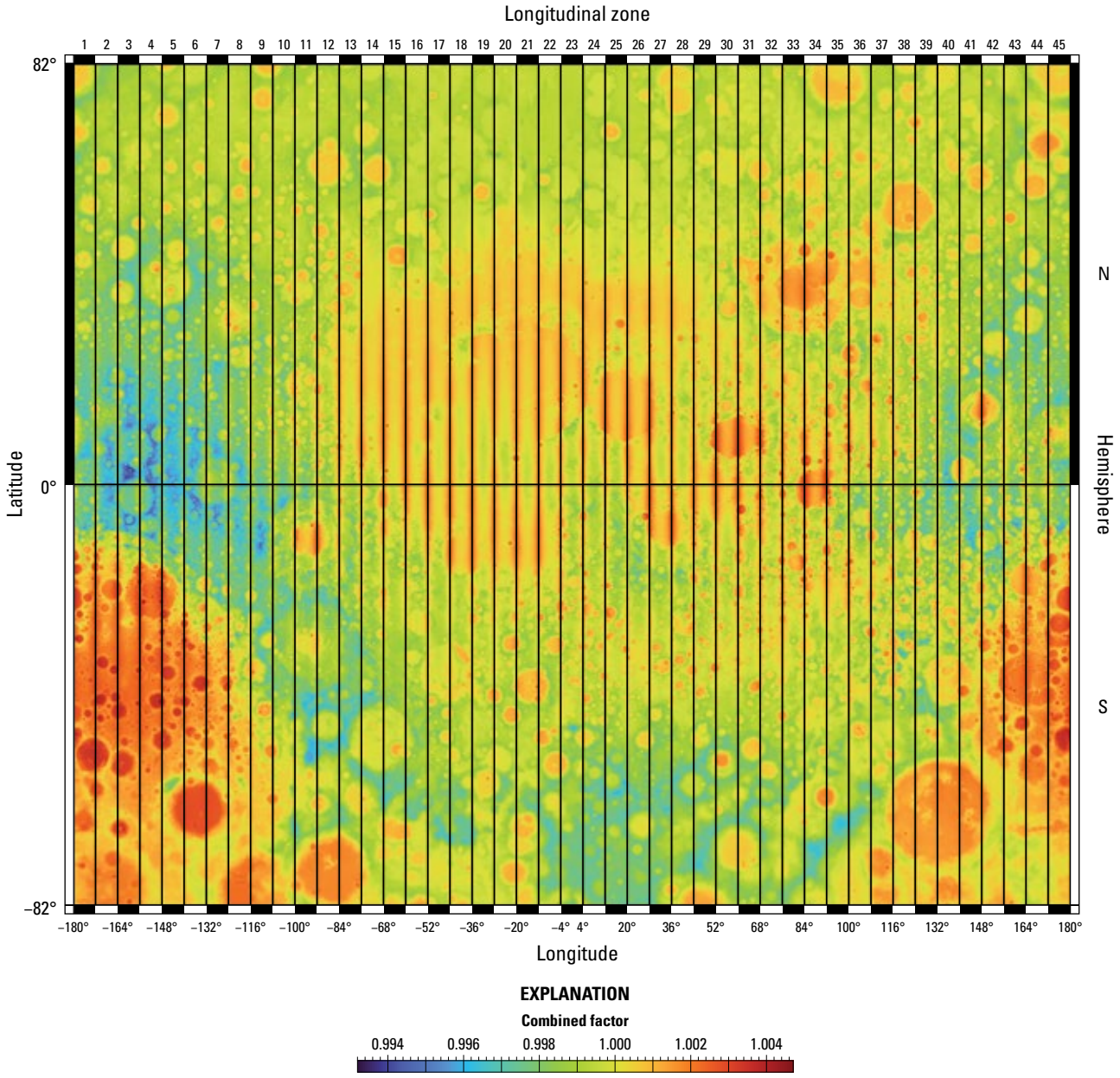


Figure 6. Map showing combined factor calculated for the transverse Mercator projections of the 90 zones in the Lunar Transverse Mercator (LTM) system. Calculation based on the grid scale factor and the height factor (see figs. 4, 5). See tables 2 and 3 for east–west and north–south extents of the LTM zones.

Polar Stereographic and Lunar Polar Stereographic (LPS) System

This section outlines the specifications for the LPS system for polar regions of the lunar surface. The LPS system would be used in tandem with the LTM system much like the UPS system is used in tandem with the UTM system (NGA, 2014b). LPS utilizes a polar stereographic projection for the north and south poles of the Moon. Stereographic projections are azimuthal, planar with no associated projection curvature, and conformal (Snyder, 1987; NGA, 2014b) and great circles appear as straight lines. See Snyder (1987) for discussion about the properties of this map projection.

A polar stereographic system is especially well suited for navigation in high latitudes (for example, the Arctic and Antarctic regions) (Snyder, 1987; Karney, 2011, NGA, 2014a). We note that oblique stereographic map projections are not utilized in this work; the only stereographic projections used are centered on the lunar poles.

Definition of Polar Stereographic and The LPS System

There are many ways to convert planetocentric coordinates to polar stereographic coordinates. Unlike LTM, where Karney-Krüger equations must be utilized (Karney, 2011), no specific set of equations needs to be utilized to recreate the LPS system. Similarly to the LTM system, the LPS system uses a small number of agreed-upon concepts and set parameters to define projection areas and the conversion. We use Snyder's (1987) spherical equations for the polar stereographic projection and the LPS system. In the software associated with this work, the Snyder (1987) ellipsoidal equations for polar stereographic are also provided. The USGS recommends the continued use of these equations for correct positioning (McClernan and others, 2023); however, other equations that compute polar stereographic projections can reproduce the results of this work. We detail the parameters used in our implementation of the polar stereographic projection in this section.

Polar Stereographic and LPS Coordinate Structure

The polar stereographic projection has a similar coordinate structure to the transverse Mercator projection. However, polar stereographic coordinates do not need to specify both a zone and a hemisphere as each hemisphere is represented by a single zone. We refer to the hemispheres as the north and south LPS zones.

A conversion from latitude, φ, and longitude, λ, to polar stereographic coordinates produces the rectangular coordinates of hemisphere, *H*; easting, *E*; and northing, *N*. The grid scale factor and grid convergence can also be computed although, Snyder (1987) does not provide equations for grid convergence. When converting from LPS back to latitude and longitude, the hemisphere will need to be provided to properly complete the inverse conversion.

Field communication of position is identical to LTM. Numerical values of easting and northing are reported as a coordinate pair. Easting is always reported prior to northing or "easting" and "northing" must be stated after the corresponding LPS zone coordinate value.

Polar Stereographic and LPS Zones

Polar stereographic has two natural zones, based on the north polar stereographic and south polar stereographic projections (see figs. 7, 8). These zones are referenced with a point hemisphere location: that is, *H* = "S" or "N". A specific zonal reference, *Z*, is unneeded as there are only two projections; however, these areas are referred to as the "north polar zone" and the "south polar zone".

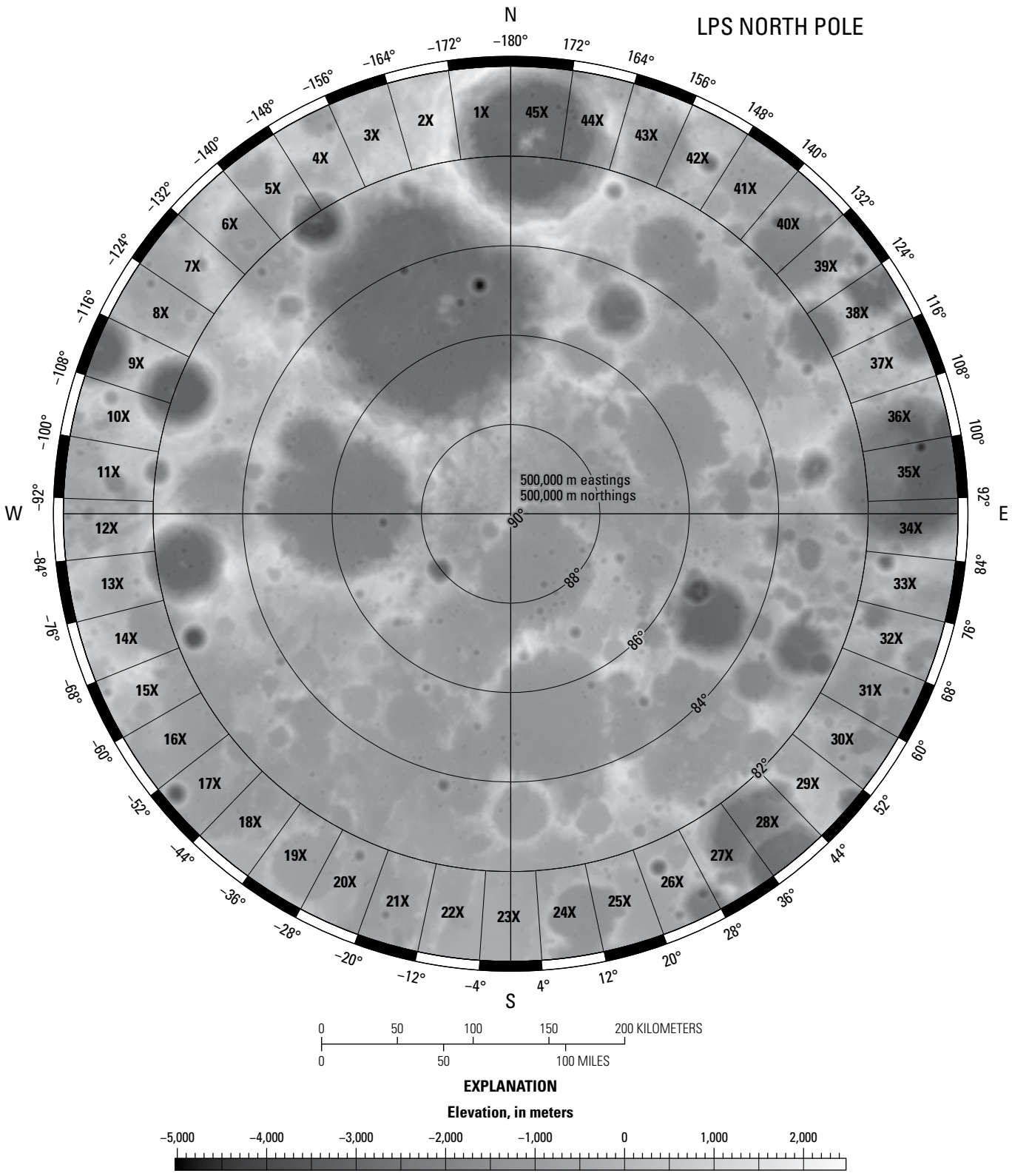


Figure 7. Map showing the Lunar Polar Stereographic (LPS) system for the north pole of the Moon. Elevation from a Lunar Orbiter Laser Altimeter (LOLA) 118-meter (m)-resolution digital elevation model (LOLA Science Team, 2021). Lunar Transverse Mercator (LTM) zones in the extended LTM range are shown; however, this is not in the primary LTM usage range.

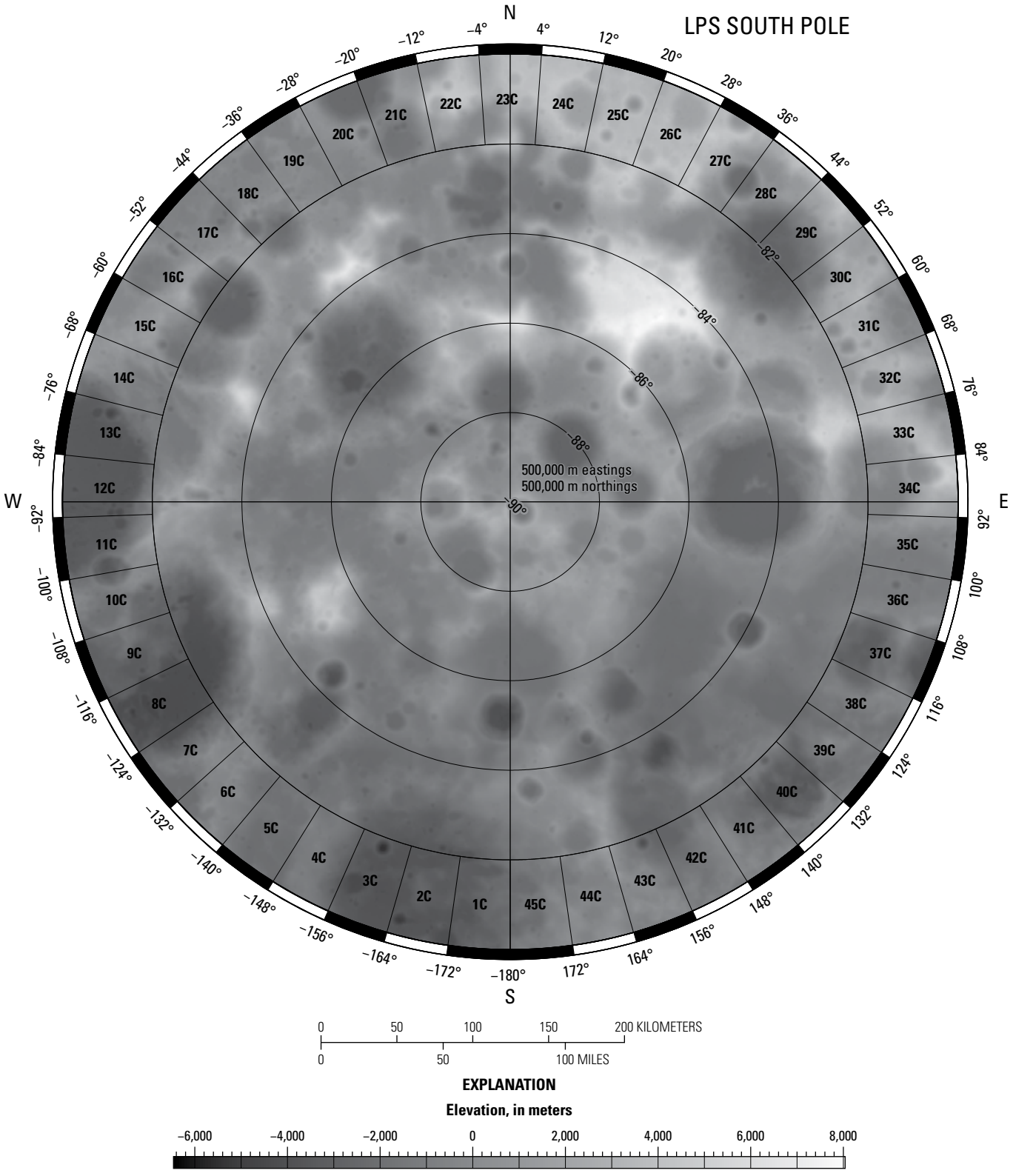


Figure 8. Map showing the Lunar Polar Stereographic (LPS) system for the south pole of the Moon. Elevation from a Lunar Orbiter Laser Altimeter (LOLA) 118-meter (m)-resolution digital elevation model (LOLA Science Team, 2021). Lunar Transverse Mercator (LTM) zones in the extended LTM range are shown; however, this is not in the primary LTM usage range.

Polar Stereographic and LPS Projection Range

The polar stereographic map projection and the LPS system are designed for use in latitudes north of 80° and south of −80°. At latitudes between −80° and 80°, LTM should be used. To allow for some flexibility between mapping with LPS and LTM, the LPS system overlaps with the extended range of the LTM system between −82° and −80° and between 80° and 82°. See the "Transverse Mercator and Lunar Transverse Mercator (LTM) System" section for more information.

When measured from the pole, the LPS polar stereographic projections have a usable distance of 10° or a 20° (~303 km) distance measured across the entire projection area. As mentioned, the transverse Mercator projection is limited based on surface distortion and scale error. Polar stereographic projection has similar distortion and scale error issues; however, the scale error of stereographic projections is allowed to be somewhat greater than the scale error of transverse Mercator projections as there are no alternative conformal projections that function well at high latitudes.

Polar Stereographic and LPS Projection Axis

The polar stereographic projection axis is not a linear axis across the map area as it is for a transverse Mercator projection; it is a single point of minimum scale centered at the projection origin (Snyder, 1987; Dennis, 2022). For the lunar north-polar stereographic projection, the projection axis is defined at the lunar north pole as

$$\varphi_0, \lambda_0 = \{90°, 0°\},$$

where
φ_0, λ_0 is the latitude and longitude of the polar stereographic projection origin.
Similarly, for the lunar south-polar stereographic projection, the projection axis is defined as

$$\varphi_0, \lambda_0 = \{-90°, 0°\}.$$

These projection origins are also utilized for the LPS north and south polar zones.

Polar Stereographic and LPS Central Scale Factor

To support LPS and its associated polar stereographic map projections north of 80° and south of −80°; we have chosen a central scale factor similar to that of the UPS system to provide a scale error similar to the Earth's (Snyder, 1987; NGA 2014b). We set the central scale factor to 0.994 (exactly), or −6 m/km at the projection center, using double floating-point precision values for both LPS projections. The projections are considered "true scale" at ±81° where the grid scale factor is 1.0. We discuss the implications to map distortion in later sections.

Scale errors in a polar stereographic projection are generally high as the lunar sphere is being projected onto a flat plane tangent to the reference surface of the Moon (Snyder, 1987). The projection has no associated curvature which causes the projection scale error to accumulate across relatively short surface distances. As such, scale error and, thus, linear distortion accumulates in all orientations with increasing distance from the poles to equatorial regions. It is also not possible to achieve a ±1:1,000 scale error at the lunar poles without the development and use of an LDP or without limiting the polar stereographic projection range to about 5° from the poles. This finding has been reported to the AGDT and has been accepted to finalize a working model of LPS (Flight Operations Directorate, NASA, oral commun., 2024).

Polar Stereographic and LPS False Origins

Similarly to transverse Mercator and LTM, false origins have never been applied to a lunar polar stereographic map projection (Hare and Malapert, 2021). The LPS system applies false origins in its coordinate system to shift the central $\{0, 0\}$ point from the north and south poles to the lower left-hand corner of each projection area. As the LPS projections cover equal areas in both the *x*- and *y*-dimensions, the false origins are the same. For both the lunar south and north pole, the LPS system uses a false easting and a false northing of 500 km (McClernan and others, 2023). These false origins were selected to keep all LPS coordinate values positive without changing the number of significant digits. We also note that 500 km is one quarter of the UPS false origins of 2,000 km (NGA, 2014b). False origins are recommended and needed in the LPS system when used for navigation as the arithmetic of LGRS requires positive coordinate values.

Polar Stereographic Grid Directions

When navigating with a polar stereographic map, the geographic pole is in the center of the map projection and when orienting the map for navigational purposes the use of a consistent grid north is recommended. For north polar stereographic projection, grid North is aligned in the direction of the 180° meridian. For the south polar stereographic projection, grid north is oriented in the direction of the 0° meridian. Grid east, south, and west are defined in angular distances of 90° from this designation of grid north. Using the JPL DE 421 ME frame aligns the lunar 0° prime meridian to face Earth. Thus, on the lunar south pole grid north may be approximately determined by orienting the map to face Earth.

LPS System Conversion Equations

In this section, we describe the polar stereographic map projection equations as implemented by Snyder (1987) and discuss the parameters utilized. These equations only apply to polar stereographic map projections. Oblique stereographic projections are beyond the scope of this document.

LPS Parameters and Formulization

LPS utilizes a small number of previously introduced parameters to assure interoperability and proper positioning. The parameters are listed in table 5 with their associated symbols, values, and units. These values define the north and south polar LPS zones while only differing in the projection origin latitude.

Table 5. Lunar Polar Stereographic (LPS) system parameters.

Parameter	Symbol	Value	Units
North polar zone (see fig. 7)			
Central scale factor	k_0	0.994	Unitless
Latitude of projection origin	φ_0	90	Arc-degrees
Longitude of projection origin	λ_0	0	Arc-degrees
False easting	F_E	500,000	Meters
False northing	F_N	500,000	Meters
Lunar radius	a	1,737,400	Meters
South polar zone (see fig. 8)			
Central scale factor	k_0	0.994	Unitless
Latitude of projection origin	φ_0	–90	Arc-degrees
Longitude of projection origin	λ_0	0	Arc-degrees
false easting	F_E	500,000	Meters
false northing	F_N	500,000	Meters
lunar radius	a	1,737,400	Meters

Forward Conversion $\lambda, \varphi \rightarrow \{E, N\}$

This section outlines the steps used to convert a point location from planetocentric latitude and longitude to the LPS grid. We refer to the forward conversion to LPS with the following general mapping statement:

$$LatLon : \left(\varphi, \lambda, \varphi_0, \lambda_0, a, k\right) \overset{LPS}{\rightarrow} \{E, N\}, \tag{64}$$

where the latitude, φ, and longitude, λ, are converted to the LPS rectangular coordinates of easting, *E*, and northing, *N*, for the associated polar hemisphere, *H*. The parameters in the map projection used in this calculation are φ_0, λ_0, a, k, F_E, and F_N, where all angular measurements are converted to radians. The parameters are discussed in previous sections.

The general polar stereographic equations from Snyder (1987) can be used to calculate the easting and northing coordinates, *E* and *N*:

$$E = ak\left(\cos\left(\varphi\right)\sin\left(\lambda - \lambda_0\right)\right) + F_E, \tag{65}$$

and

$$N = ak\left(\cos\left(\varphi_0\right)\sin\left(\varphi\right) - \sin\left(\varphi_0\right)\cos\left(\varphi\right)\cos\left(\lambda - \lambda_0\right)\right) + F_N, \tag{66}$$

where the grid scale factor, k, is calculated as

$$k = 2k_0 / \left(1 + \sin\left(\varphi_0\right)\sin\left(\varphi\right) + \cos\left(\varphi_0\right)\cos\left(\varphi\right)\cos\left(\lambda - \lambda_0\right)\right). \tag{67}$$

For a more direct conversion process Snyder's (1987) equations can be simplified for each pole. We have implemented two versions of these equations for each pole. To determine which polar area a potential point would be located in, we use a simple query of φ_0 and use it as a processing flag as follows.

If $\varphi_0 \equiv 90°$, the equations are simplified for calculating a north-polar stereographic projection:

$$E = 2ak_0\left(\tan\left(\frac{\pi}{4} + \frac{\varphi}{2}\right)\sin\left(\lambda - \lambda_0\right)\right) + F_E, \tag{68}$$

$$N = 2ak_0\left(\tan\left(\frac{\pi}{4} + \frac{\varphi}{2}\right)\cos\left(\lambda - \lambda_0\right)\right) + F_N, \tag{69}$$

and

$$k = 2k_0 / \left(1 + \sin\left(\varphi\right)\right). \tag{70}$$

If $\varphi_0 \equiv -90°$, the equations are simplified for calculating a south-polar stereographic projection:

$$x = 2ak_0\left(\tan\left(\frac{\pi}{4} - \frac{\varphi}{2}\right)\sin\left(\lambda - \lambda_0\right)\right), \tag{71}$$

$$y = -2ak_0\left(\tan\left(\frac{\pi}{4} - \frac{\varphi}{2}\right)\cos\left(\lambda - \lambda_0\right)\right), \tag{72}$$

and

$$k = 2k_0 / \left(1 - \sin\left(\varphi\right)\right). \tag{73}$$

x and y are rectangular coordinates of the polar stereographic map projection. Applying a false origin shifts x and y to the final easting and northing position, that is, $E = x + F_E$ and $N = y + F_N$.

Generally, these equations are only valid if $\varphi \neq \varphi_0$, thus if the point being converted is at the same location as the stereographic projection axis, the point cannot be plotted or converted between formats. This is a current restriction in Snyder's (1987) spherical equations. In the software release associated with this document, we have accounted for this by explicitly assigning the value of φ if $\varphi = \varphi_0$. Additionally, in the provided coordinate conversion software the Snyder (1987) equations for a stereographic projection on an ellipsoid are also provided, which do not have critical point limitations as in the spheroidal version.

We also note that the ellipsoidal version of Snyder's (1987) equations can be utilized to process points on the sphere; however, the ellipsoidal equations require modification and the eccentricity, e, and flattening, f, terms must be set to zero, as the LRS uses a sphere with a uniform radius.

Inverse Conversion $E, N \rightarrow \{\lambda, \varphi\}$

This section outlines the steps used to convert a point location from the LPS grid to planetocentric latitude and longitude. We refer to the inverse conversion from LPS coordinates to latitude and longitude with the following general mapping statement:

$$LPS:\left(E, N, \varphi_0, \lambda_0, a, k\right) \overset{LatLon}{\rightarrow} \{\varphi, \lambda\}. \tag{74}$$

The parameters used during the forward conversion, φ_0, λ_0, a, k, F_E, and F_N, are used during the inverse conversions. Similarly to the forward conversion, equations to complete this transformation are from Snyder (1987). To compute latitude (φ) and longitude (λ), use the following equations:

$$\varphi = \sin^{-1}\left(\cos(c)\sin(\varphi_0) + \left(Y\sin(c)\cos(\varphi_0)\right)/\rho\right), \tag{75}$$

and

$$\lambda = \lambda_0 + \tan 2^{-1}\left(X\sin(c)/\left(\rho\cos(\varphi_0)\cos(c) - X\sin(\varphi_0)\sin(c)\right)\right). \tag{76}$$

$\tan 2^{-1}$ is the arctangent with four quadrant correction, and
ρ is the planar distance from the center of the polar stereographic projection, defined as

$$\rho = \sqrt{x^2 + y^2}, \tag{77}$$

where

x is a rectangular coordinate with the false easting removed, $x = \left(E - F_E\right)$, and
y is a rectangular coordinate with the false northing removed, $y = \left(N - F_N\right)$.
c is the geodesic distance on a sphere defined as

$$c = 2tan2^{-1}\left(\rho/\left(2ak_0\right)\right). \tag{78}$$

If $\varphi_0 = 90°$ and all points are on the north pole, then longitude can be determined from the following simplified equation:

$$\lambda = \lambda_0 + \tan 2^{-1}\left(x/y\right). \tag{79}$$

If $\varphi_0 = -90°$ and all points are on the south pole, then longitude can be determined from the following simplified equation:

$$\lambda = \lambda_0 + \tan 2^{-1}\left(x/-y\right). \tag{80}$$

The resulting coordinate of $\{\lambda, \varphi\}$ is in radians and can be converted to degrees to determine the planetocentric coordinate. Snyder's (1987) ellipsoidal equations can also be utilized to complete this conversion with similar modifications required as mentioned in the forward conversion.

LPS System Sample Conversion and Grid Generation Program

Preliminary software to complete the forward and inverse LPS system coordinate conversions is provided in appendix 1. Additional equations to create shapefiles of the LPS system are provided in appendix 2.

LPS Map Projection and Sample WKT

Here we include sample WKT for the LPS north and south poles (figs. 7, 8). These projections are not included in Esri's ArcGIS, QGIS, the EPSG registry, or the IAU projection library (Hare and Malapert, 2021); however, this text can be copied and pasted into most GIS software projection libraries to reproduce the LPS projections.

WKT Sample for LPS North Pole

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / North Polar”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“North Polar”,
    METHOD[“Polar Stereographic (variant A)”,
      ID[“EPSG”,9810]],
    PARAMETER[“Latitude of natural origin”,90,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.994,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
ID[“USGS”,7190092,N]]]
```

WKT Sample for LPS South Pole

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / South Polar”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
```

```
            LENGTHUNIT[“metre”,1]]],
      PRIMEM[“Reference Meridian”,0,
        ANGLEUNIT[“degree”,0.0174532925199433]],
      ID[“IAU”,30100,2015]],
   CONVERSION[“South Polar”,
      METHOD[“Polar Stereographic (variant A)”,
        ID[“EPSG”,9810]],
      PARAMETER[“Latitude of natural origin”,-90,
        ANGLEUNIT[“degree”,0.0174532925199433],
        ID[“EPSG”,8801]],
      PARAMETER[“Longitude of natural origin”,0,
        ANGLEUNIT[“degree”,0.0174532925199433],
        ID[“EPSG”,8802]],
      PARAMETER[“Scale factor at natural origin”,.994,
        SCALEUNIT[“unity”,1],
        ID[“EPSG”,8805]],
      PARAMETER[“False easting”,500000,
        LENGTHUNIT[“metre”,1],
        ID[“EPSG”,8806]],
      PARAMETER[“False northing”,500000,
        LENGTHUNIT[“metre”,1],
        ID[“EPSG”,8807]]],
   CS[Cartesian,2],
      AXIS[“(E)”,east,
        ORDER[1],
        LENGTHUNIT[“metre”,1]],
      AXIS[“(N)”,north,
        ORDER[2],
        LENGTHUNIT[“metre”,1]],
ID[“USGS”,7190091,S]]]
```

LPS Surface Distortion Analysis

As previously mentioned, NASA’s Flight Operations Directorate recommended to the AGDT that the projections have a maximum scale error of ±1:1,000 or ±1 m/km (McClernan and others, 2023; see recommendation 1 from the “Grid System for Navigation” section). This translates to a grid scale factor between 0.999 and 1.001. As the figures below show, achieving this target scale error is not possible for polar stereographic maps plotted from 90° to 80° or −90° to −80°. The projection zone will have a limited area unless a higher scale error is accepted. The tradeoff in projection coverage is accepting a higher scale error and distortion to support a larger area, such as the entire lunar polar area.

To complete a scale error analysis and calculate associated linear distortion of the polar stereographic projections used in LPS, we calculated the scale factor and combined factor for the north and south polar LPS zones. In geodetic surveying where precision and accuracy are prioritized, the combined factor is used to adjust distances above and below the planetary reference surface. Although calculation of map linear distortion is discussed in the “Background” section, we restate the values used in the determination of the combined factor as it applies to a lunar polar stereographic map for clarity here.

To calculate the combined factor, we utilize equations 1 and 2, using the grid scale factor, k, calculated from equation 70 or 73 (see figs. 9, 12). We utilize a LOLA 118-m-resolution gridded DEM to calculate the height factor (figs. 10, 13) (LOLA Science Team, 2021). The product of k and h_F produces the combined factor, a value that can reduce an observed topographic distance to the polar stereographic map projection surface (figs. 11, 14). We discuss the distortions of the north and south LPS zones below.

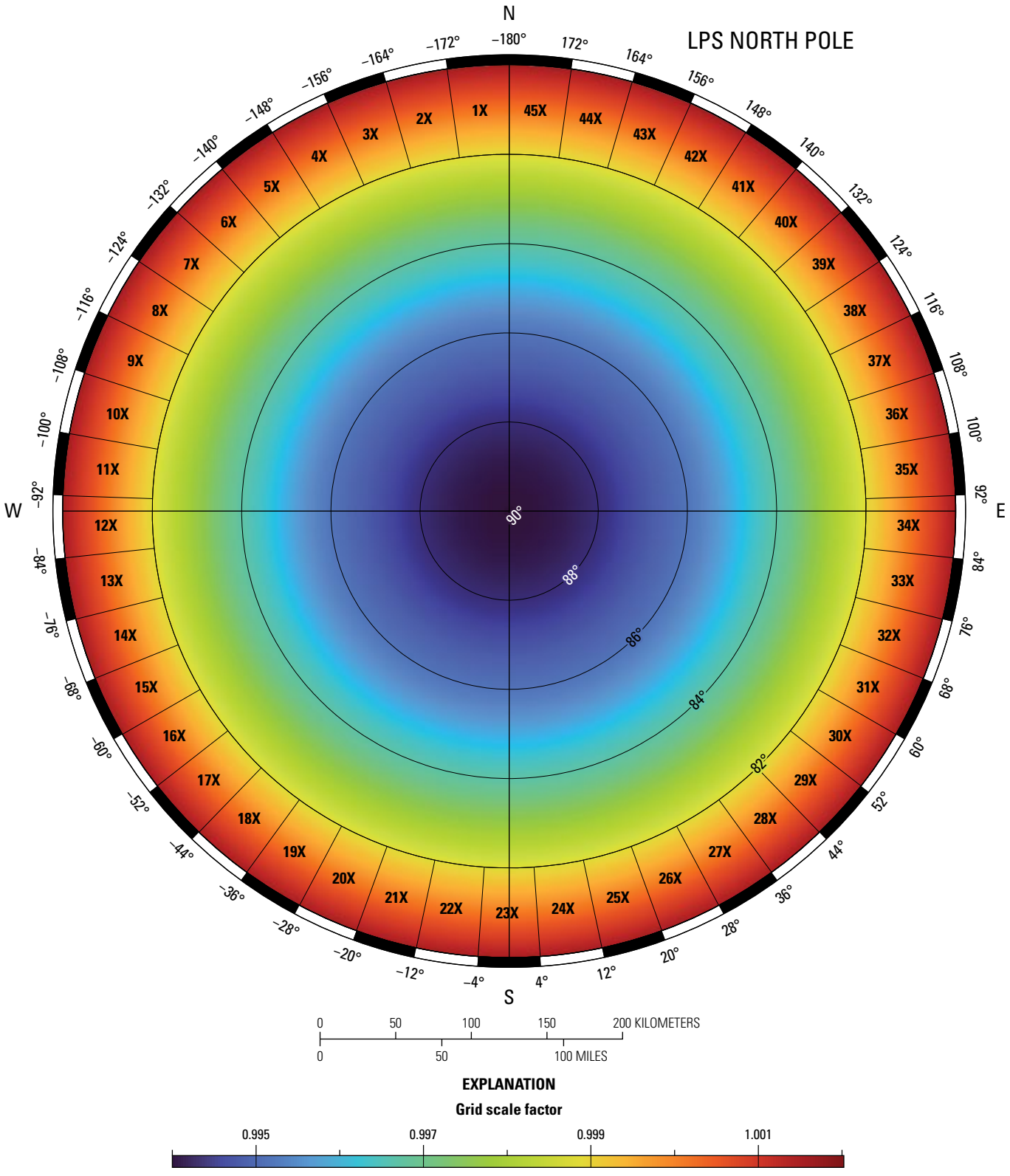


Figure 9. Map showing grid scale factor calculated for the Lunar Polar Stereographic (LPS) projection of the lunar north pole. Overlapping Lunar Transverse Mercator (LTM) zones in the extended LTM range are shown; however, this is not in the primary LTM usage range.

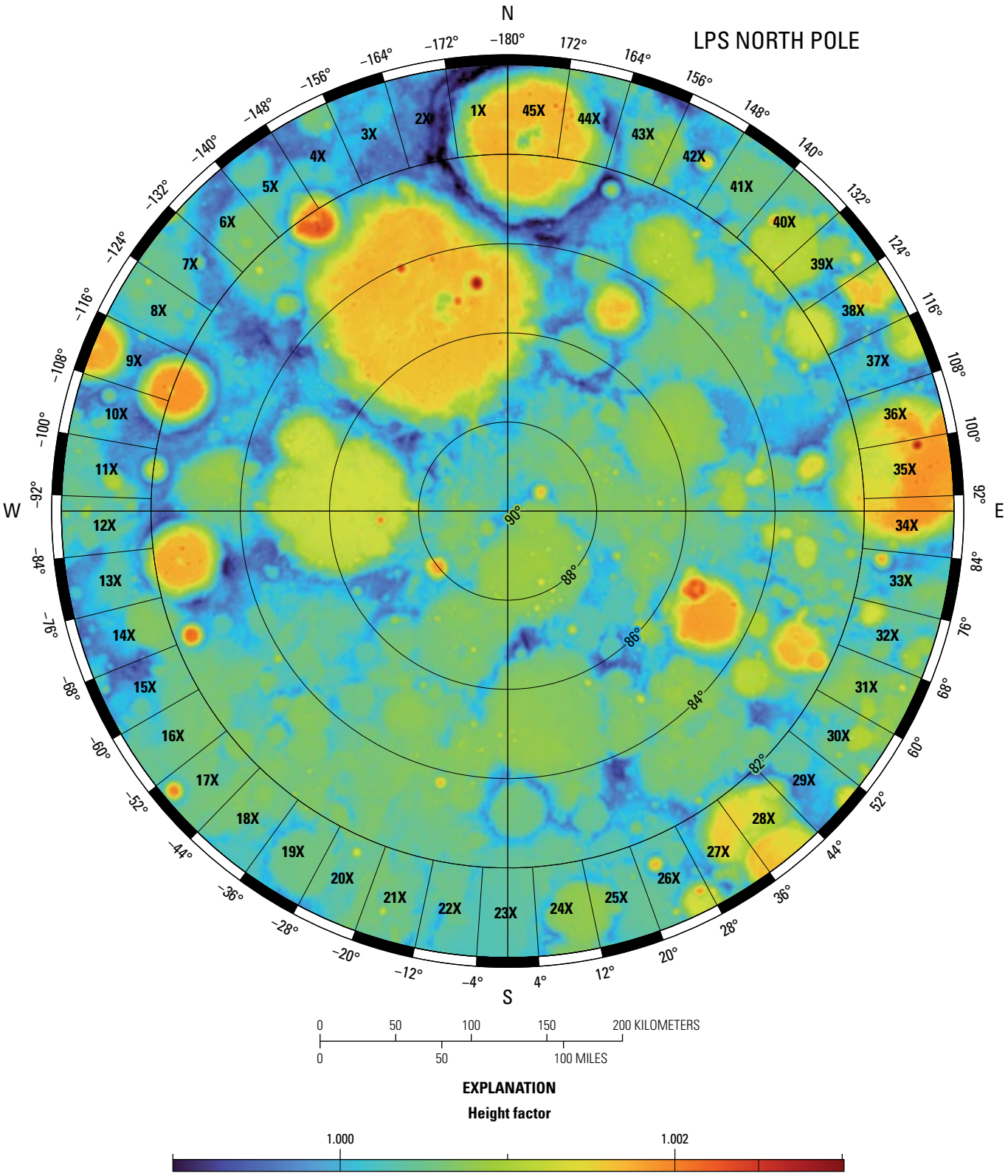


Figure 10. Map showing height factor calculated for the north pole region of the Lunar Polar Stereographic (LPS) system. Calculated from lunar elevation data from a Lunar Orbiter Laser Altimeter (LOLA) 118-meter-resolution digital elevation model (LOLA Science Team, 2021). Overlapping zones in the extended Lunar Transverse Mercator (LTM) range are shown; however, this is not in the primary LTM usage range.

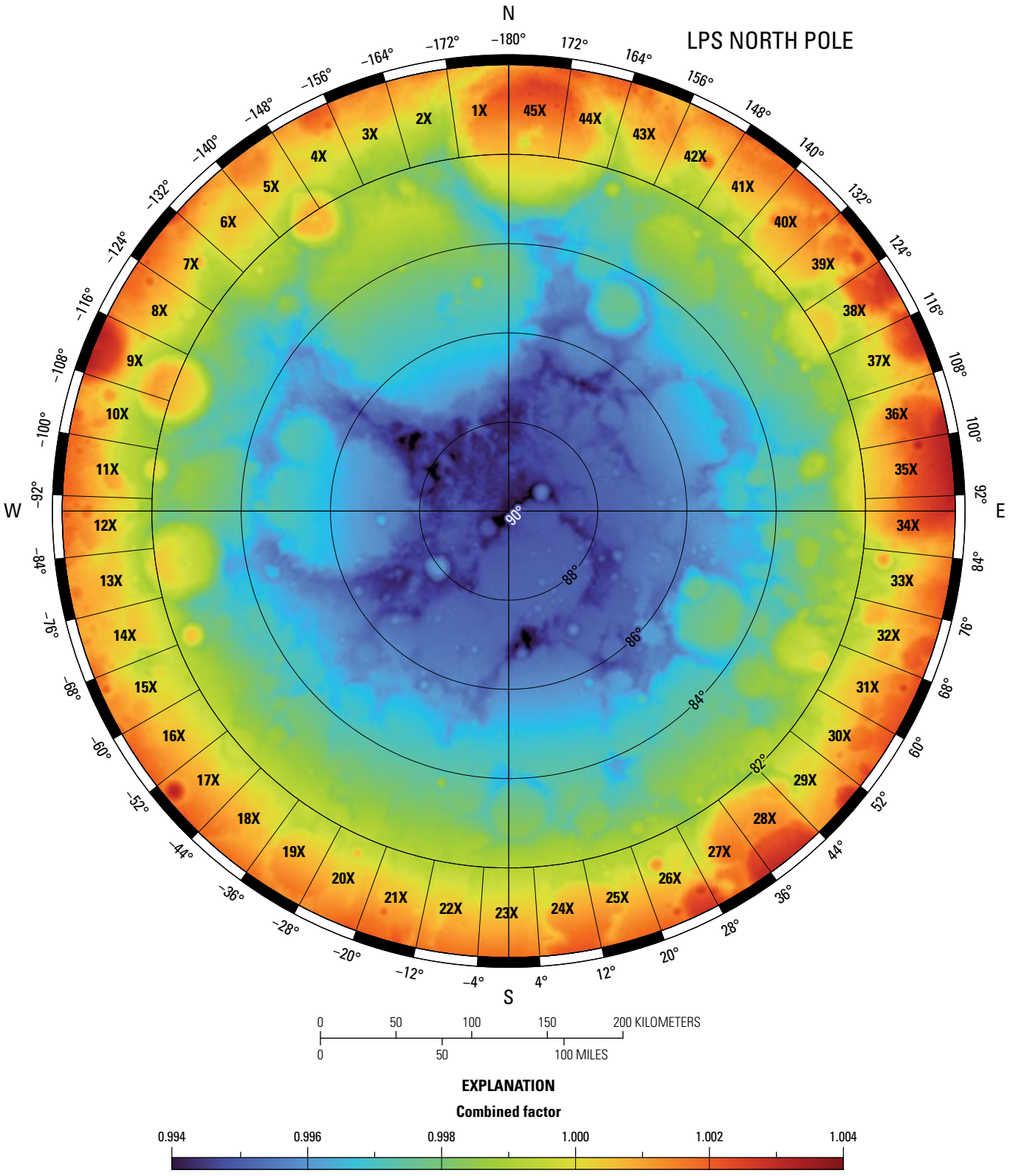


Figure 11. Map showing combined factor calculated for the Lunar Polar Stereographic (LPS) projection of the lunar north pole. Calculation based on the grid scale factor and the height factor (see figs. 9, 10). Overlapping Lunar Transverse Mercator (LTM) zones in the extended LTM range are shown; however, this is not in the primary LTM usage range.

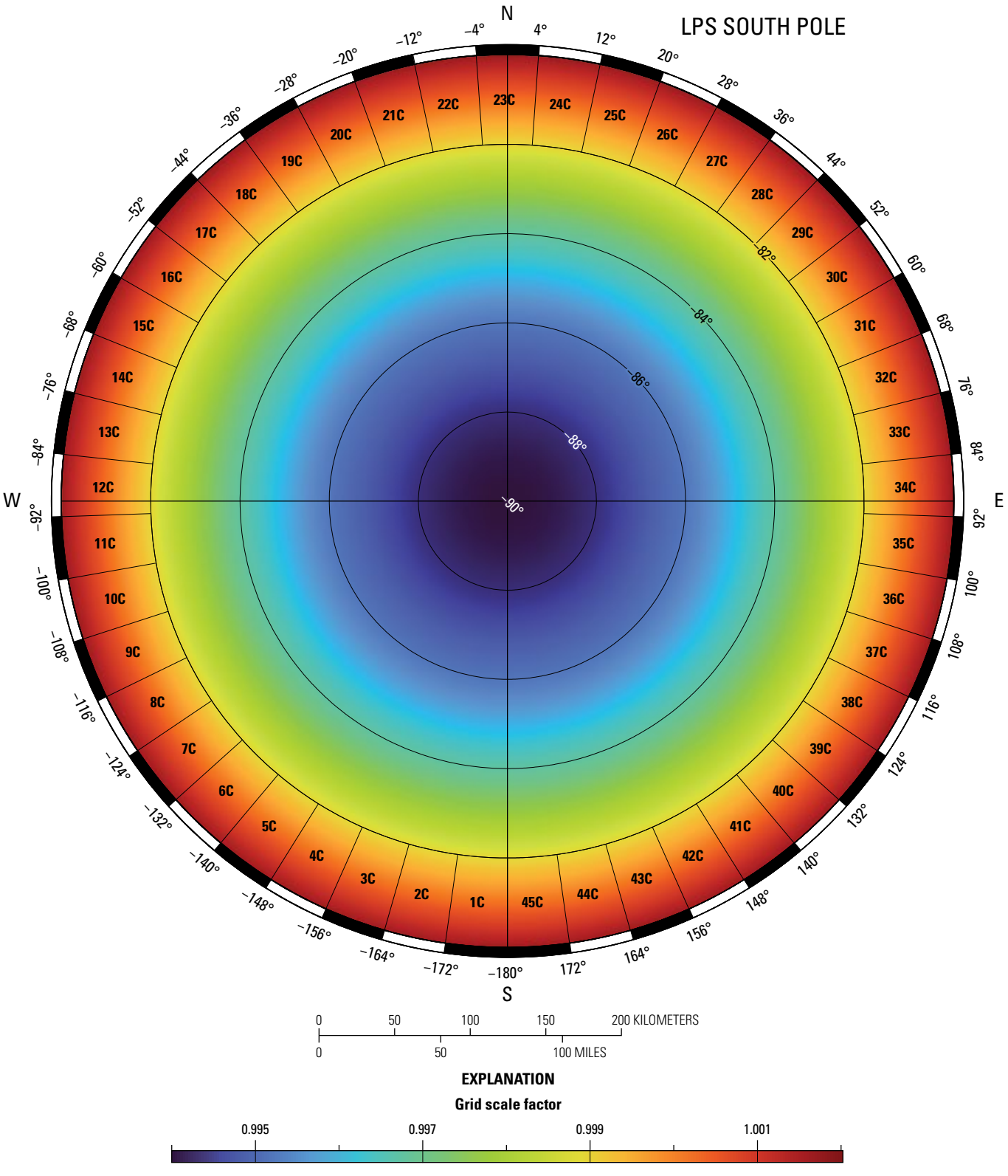


Figure 12. Map showing grid scale factor calculated for the Lunar Polar Stereographic (LPS) projection of the lunar south pole. Overlapping zones in the extended Lunar Transverse Mercator (LTM) range are shown; however, this is not in the primary LTM usage range.

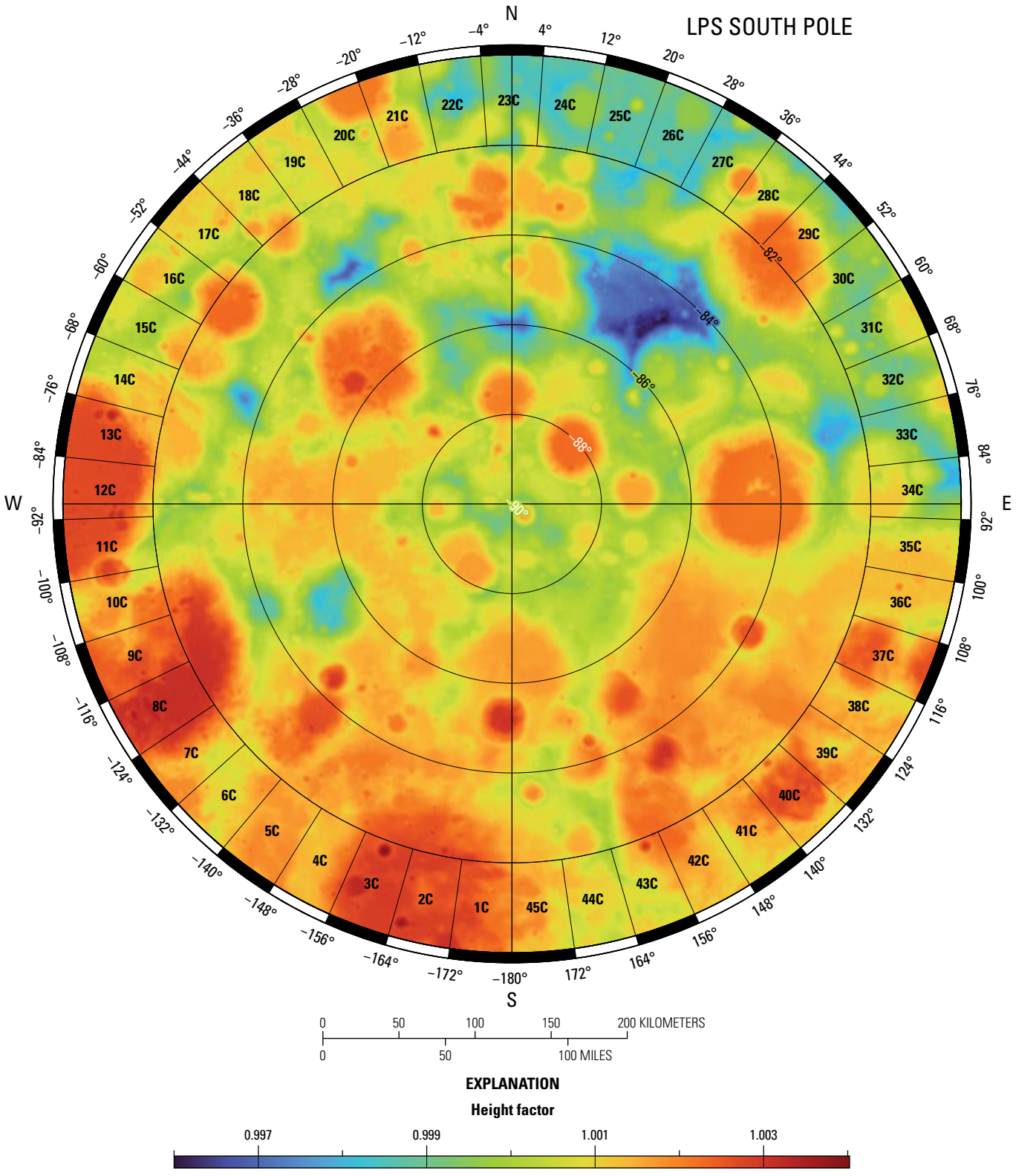


Figure 13. Map showing height factor calculated for the south pole region of the Lunar Polar Stereographic (LPS) system. Calculated from lunar elevation data from a Lunar Orbiter Laser Altimeter (LOLA) 118-meter-resolution digital elevation model (LOLA Science Team, 2021). Overlapping zones in the extended Lunar Transverse Mercator (LTM) range are shown; however, this is not in the primary LTM usage range.

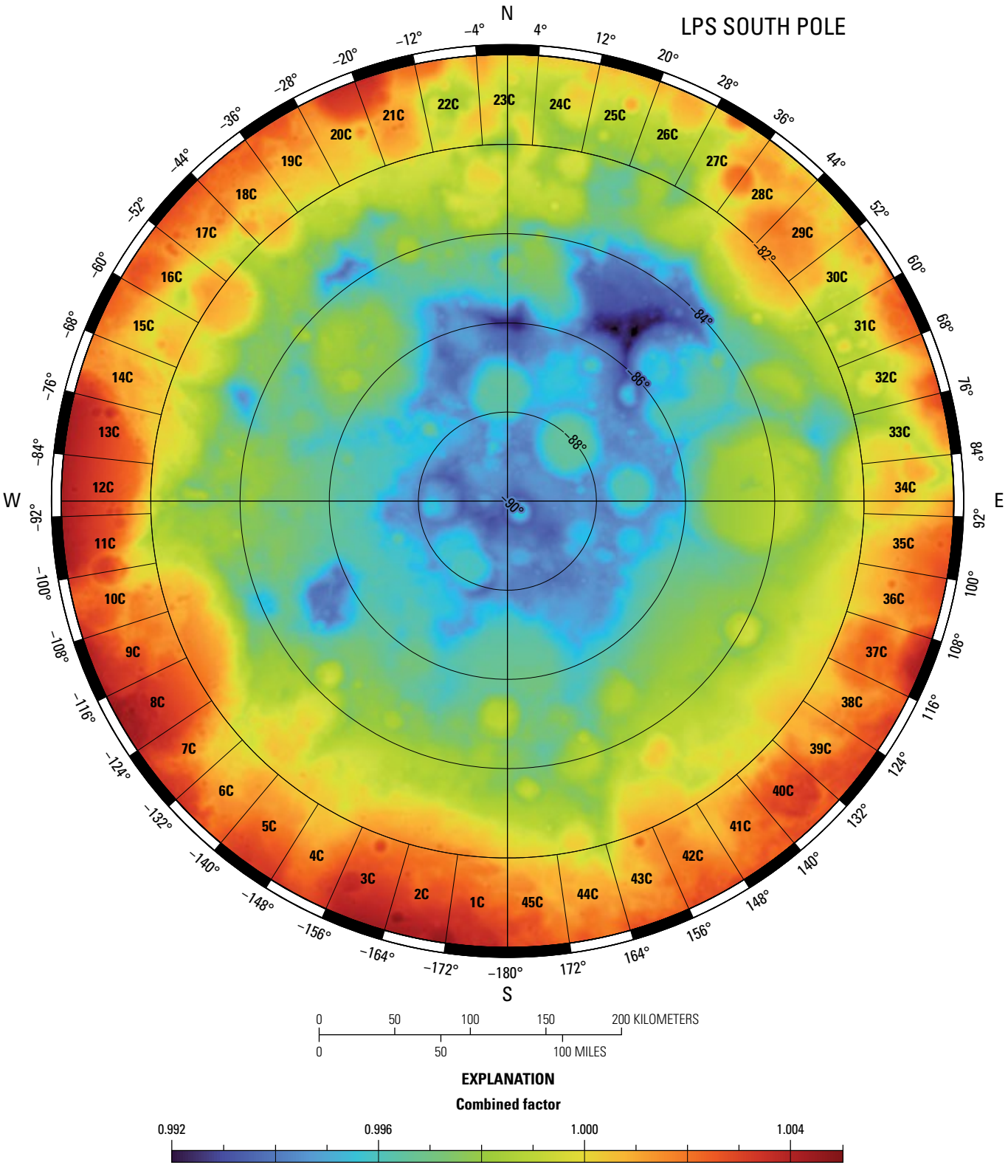


Figure 14. Map showing combined factor calculated for the Lunar Polar Stereographic (LPS) projection of the lunar south pole. Calculation based on the grid scale factor and the height factor (see figs. 12, 13). Overlapping zones in the extended Lunar Transverse Mercator (LTM) range are shown; however, this is not in the primary LTM usage range.

Lunar North Polar Stereographic Ground Distortion

Lunar North Polar Stereographic Grid Scale Factor

The grid scale factor was calculated for the LPS projection of the lunar north pole using equation 70 (fig. 9). We have selected a central scale factor of 0.994 at the projection origin (the north pole, $\{\varphi_0, \lambda_0\} = \{90°, 0°\}$), which is the same as the terrestrial central scale factor for the UPS system. The grid scale factor in conformal maps only increases in one direction from the projection axis; however, because the polar stereographic map projection has a reference point instead of an axis across the map area, the grid scale factor increases in all directions from the map projection center toward the equator (Snyder, 1987). The grid scale factor increases to 1.000 at lat 81°06′ and just greater than 1.0016 at lat 80° (fig. 9).

When comparing distances on the north polar stereographic map to the distances on the lunar reference spheroid, the distance measured from the map projection grid must be divided to calculate the corresponding distance on the reference spheroid. At the border of the projection (80°) the scale error expands distances measured on the map projection by +1.6 m/km. At the projection center there is a scale error of −6 m/km and at 81°06′ there is no scale error between distances measured on the reference surface and on the map projection surface.

Lunar North Polar Height Factor

We used a lunar sphere radius of 1,737,400 m and a LOLA 118-m-resolution gridded DEM (LOLA Science Team, 2021) to calculate the height factor using equation 2 (fig. 10). Note that the height factor is independent from the map projection. Negative topography expresses a height factor greater than 1, indicating that grid distances are longer than ground distances. Positive topography expresses a height factor less than 1, indicating that grid distances are shorter than ground distances. As shown in figure 10, most of the north pole region displays a height factor greater than 1 except for isolated peaks and crater rims. The total height factor range is 0.00455, with a minimum value of 0.998630 and a maximum value of 1.00317.

Lunar North Polar Stereographic Combined Factor

The combined factor was calculated from the height factor and grid scale factor values for a polar stereographic projection of the lunar north pole using equation 1 (fig. 11). As shown in figures 7 and 11, the topography influences the overall combined factor; however, the grid scale factor increases toward the map projection border and imposes a gradient across the entire north pole region. The total combined factor range is 0.0099448 with a minimum value of 0.993576 and a maximum value of 1.0035242. The combined factor is smallest at the map projection center and at the locations of high topography and greatest in deep craters and at the edges of the map projection border. Translated into linear distortion, expected distortion ranges from −6.4 m/km to +3.5 m/km across the map area.

Lunar South Polar Stereographic Ground Distortion

Lunar South Polar Stereographic Grid Scale Factor

The grid scale factor was calculated for the LPS projection of the lunar south pole using equation 73 (fig .12). The central scale factor is 0.994 at the projection origin (the south pole, $\{\varphi_0, \lambda_0\} = \{-90°, 0°\}$), which is the same as the LPS north polar zone and the UPS system. Similarly to the north pole, the grid scale factor for the south pole increases in all directions from the map projection center toward the equator (Snyder, 1987). Grid scale factor values are identical to the north pole and increase to 1.000 at lat 81°06′ and just greater than 1.0016 at lat 80° (fig .12).

At the border of the projection (80°) the scale error is +1.6 m/km, at the projection center there is a scale error of −6 m/km, and at 81°06′ there is no scale error between distances measured on the reference surface and on the map projection surface. As discussed previously, longitude does not affect the distortion (fig. 12).

Lunar South Polar Height Factor

We used a lunar sphere radius of 1,737,400 m and a LOLA 118-m-resolution gridded DEM (LOLA Science Team, 2021) to calculate the height factor using equation 2 (fig. 13). As shown in figure 13, most of the south pole region displays a height factor greater than 1 except for high-elevation mountainous regions and prominent peaks and crater rims. In general, the south pole of the Moon, which contains the margins of the about 2,500-km-wide South Pole–Aitken basin, shows greater topographic relief and is more mountainous than the north pole (figs. 7, 8). The total height factor range is 0.0090087, with a minimum value of 0.9954162 and a maximum value of 1.0044249.

The height factor values are slightly greater than values on the north pole; however, the values less than 1 on the south pole are much lower (fig. 10, 13). The large range in height factor values (fig. 13) is a result of most of the surface of the south pole being the margin of an ancient impact basin with deep craters and prominent rims superimposed, producing an especially rugged terrain (fig. 8).

Lunar South Polar Stereographic Combined Factor

The combined factor was calculated from the values of height factor and grid scale factor for a polar stereographic projection of the lunar south pole using equation 1 (fig. 14). As shown in figures 8 and 14, the topography influences the overall combined factor; however, the grid scale factor increases toward the map projection border and imposes a gradient of distortion across the entire south polar region. The total combined factor ranges from a minimum value of 0.9895 to a maximum value of 1.0048. The minimum value is from three isolated pixels (fig. 14) and most of the lower extreme values are between 0.9932 and 0.9918, occurring in the Mons Mouton area in the northeast quadrant of the south polar region (fig. 14). Three pixels have lower combined factor values. If these extreme values are removed and 0.9918 is selected as the minimum value, the combined factor range is 0.0153. The combined factor is smallest at the map projection center and at the locations of high topography and greatest in deep craters and at the edges of the map projection border. Translated into linear distortion, expected distortion ranges from −8.2 m/km to +4.8 m/km across the map area. We note these are extreme values; however, given the variation of the lunar south pole topography, these values are expected for the LPS south polar zone.

Lunar Grid Systems

This section discusses a grid system designed for lunar travel and a methodology to condense the coordinate position. The LGRS is a grid system that specifies a point or area location on the lunar surface and is designed much like its terrestrial MGRS counterpart. Additionally, LGRS coordinates can be condensed using the ACC system to report a relative position with six characters, to meet recommendation 2 from NASA's Flight Operations Directorate to the AGDT (McClernan and others, 2023). To condense LGRS coordinates, additional grids are dimensioned within the LGRS grid structure to condense LGRS coordinates to the ACC format. We describe the technical aspects of the grid in the following sections.

Lunar Grid Reference System (LGRS)

The LGRS is a globalized grid system for lunar navigation supported by the LTM and LPS projections. LGRS provides an alphanumeric grid coordinate structure for both the LTM and LPS systems. This labeling structure is utilized in a similar manner to MGRS (NGA, 2014b). The LGRS grid is defined by square areas that are 25 km × 25 km and is dimensioned based on the coordinate values of LTM and LPS systems. We refer to the grid as the LGRS 25-km grid, the areas it specifies as grid areas or 25-km-grid areas, and the map label used to refer to a grid area as a grid area designator.

LGRS Coordinate Structure for the LTM Portion

The general format of the LTM portion of LGRS is a sequence of numbers and letters that are used to define areas of the lunar surface. An example of an LGRS coordinate in the LTM portion is as follows:

35JFJ1271112229 (LGRS coordinate specifying a grid area of 1 m × 1 m)

(lower left corner at lat −30.13048481°, long 96.48515138°).

From left to right the coordinate is formatted as

I. A 1- or 2-digit integer that designates the LTM zone, an interval of longitude referred to as a "longitudinal band."

II. A letter of the alphabet between C and X that designates an interval of latitude referred to as a "latitudinal band."

III. A two-letter string that designates the 25-km-grid area. The first letter specifies the easting area designator; the second letter specifies the northing area designator.

IV. A 1- to 5-digit number that represents the easting coordinate to the desired precision. Coordinates are referenced to the lower left corner of the 25-km-grid area.

V. A 1- to 5-digit number that represents the northing coordinate to the desired precision. Coordinates are referenced to the lower left corner of the 25-km-grid area.

Letters I and O are omitted in LGRS. These coordinates are not fixed in width, as the coordinate length specifies the precision. They are written with no spaces or delimiters. LGRS and associated coordinates are designed with the capability for reporting a relative precision to specify a specific area of variable size. Zero padding the front of parts IV and V of the coordinate may be needed to specify the correct precision of a grid coordinate. The sample code provided in the Appendices coordinates parts IV and V requires a 5 digit value with zero-padding applied to convert position values correctly.

LGRS Grid Scheme for the LTM Portion

This section outlines the structure of coordinates in the LGRS LTM portion and extended range of the LTM system (lat −80° to 80° and lat −82° to 82°, respectively). The coordinate scheme designates the longitudinal band, latitudinal band and 25-km-grid area, as described below.

Longitudinal Band Number

LGRS uses the same 8°-wide longitudinal zones and numbering scheme from the LTM system. Each zone uses a 1- or 2-digit integer that designates the longitudinal band, which is part I of an LGRS coordinate in the LTM portion. See the "LTM Zone Extents" section and tables 2 and 3.

Latitudinal Band Letter

LGRS further subdivides the Moon into 8°-wide latitudinal bands from lat −80° to 80° and each band is assigned a letter from C to X, moving from south to north. Letters I and O are omitted. This is part II of an LGRS coordinate in the LTM range. This is the same latitudinal-band structure as the MGRS (NGA, 2014b). We note that for the extended range of the LTM system (−82° to −80° and 80° to 82°), we repeat the letters C and X. The latitudinal band can be determined from the following equation:

$$i = \lfloor \varphi / 8° \rfloor, \tag{81}$$

where

$\lfloor \ \rfloor$ represents floor division and
i is an index for table 6.

See figure 15 for the geometric layout.

Table 6. Latitudinal band designators for the Lunar Transverse Mercator part of the Lunar Grid Reference System (LGRS).

[*latBand*, LGRS 8° latitudinal band zone designator]

i	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	4	5	6	7	8	9	10
latBand	C	C	D	E	F	G	H	J	K	L	M	N	P	Q	R	S	T	U	V	W	X	X

To compute the latitudinal band for LTM coordinates using equation 81, LTM coordinates are first converted to latitude and longitude. This conversion uses the equations in the LTM "Inverse Conversion" section. Thus, to complete the forward conversion of LTM coordinates to LGRS, the inverse conversion of LTM northings to lunar latitude is required.

Note that the latitudinal band is based on angular measurements, whereas the remaining LGRS coordinates are based on topocentric Cartesian coordinates. Moving directly grid east or grid west will impose a north or south movement and change the latitudinal band while the LGRS grid northing value may not change. This aspect of LGRS is similar to its MGRS counterpart (NGA, 2014b).

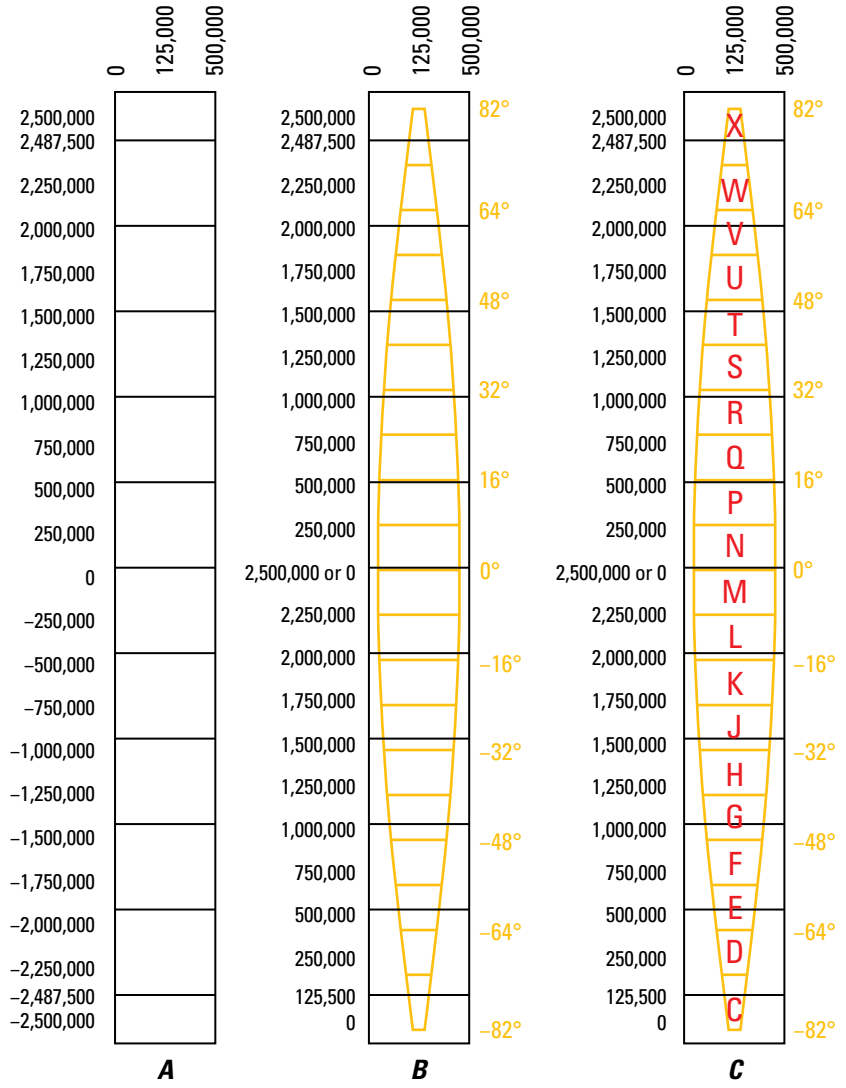


Figure 15. Schematic layout of Lunar Grid Reference System (LGRS) coordinates for an arbitrary Lunar Transverse Mercator (LTM) northern and southern hemisphere zone. *A*, LTM easting and northing coordinates for the zone (in meters). *B*, LTM 8°-wide longitudinal band subdivided into 8°-wide latitudinal bands (yellow lines). *C*, LGRS lettering scheme for 8°-wide latitudinal bands (red letters). Schematic is not to scale.

25-km-Grid Areas and Designator Scheme

LGRS uses a string of two letters to designate the 25-km-grid area, part III of an LGRS coordinate. The first letter is the easting area designator, and the second letter is the northing area designator. To determine the letters for a specific 25-km-grid area, the LTM zone and LTM coordinates need to be known and within the following limits:

$125{,}000\text{ m} \le E < 375{,}000\text{ m}$,
$0\text{ m} \le N < 2{,}487{,}500\text{ m}$ if $H =$ "N" or $12{,}500\text{ m} \le N < 2{,}500{,}000\text{ m}$ if $H =$ "S",
$1 \le Z \le 45$.

If these conditions are satisfied, the 25-km-grid area designator can be determined. The 25-km-grid easting area designator is determined using the following index function:

$$i = \lfloor E/25{,}000 \rfloor - 5, \tag{82}$$

where

E is the LTM easting coordinate,
i is an index for table 7 that references a letter from A through K, omitting I, and
5 is used to adjust the index to the correct position and “center” the grid.

Table 7. 25-kilometer-grid easting area designators for the Lunar Transverse Mercator range of the Lunar Grid Reference System (LGRS).

[*E*25*k*, LGRS 25-kilometer-grid easting area designator]

i	1	2	3	4	5	6	7	8	9	10
*E*25*k*	A	B	C	D	E	F	G	H	J	K

Unlike MGRS, LGRS does not change the easting area designator and thus, this letter structure repeats in every zone. This was done to simplify the letter structure for lunar-surface navigators. If there is a coordinate shift to the left side or right side of table 7, the navigator will know that they have switched LTM zones. Given that there is one letterset, the 25-km grid area near lat ±78° simplifies as the easting area designator is either E or F. This places a larger emphasis on knowing the direction of travel. Additionally, one can use the northing area designator, as this will shift between zones and provide a way to further delineate grid areas.

The northing LGRS 25-km-grid area designator uses three unique lettersets that shift between each LTM zone. This was implemented so that a northing area designator is not located near an identical value in an adjacent LGRS zone. Three unique lettersets are needed to avoid grid label placement near a copy of itself. This is due to the use of 8° zones in the LTM system, which results in an odd number of zones (45) in the LGRS global longitudinal reference grid. To determine the correct letterset from the LTM zone, *Z*, use the following equation:

$$i = Z \,\%\, 3, \tag{83}$$

where

% is the modulus,

3 represents the three unique lettersets used as a northing area designator.

If $Z \,\%\, 3 = 0$, see table 8; if $Z \,\%\, 3 = 1$, see table 9; and if $Z \,\%\, 3 = 2$, see table 10. Then determine the northing area designator using the following index function:

$$i = \left\lfloor \left(Y / 25{,}000 \right) \right\rfloor \,\%\, 20, \tag{84}$$

where

Y is the LTM northing coordinate,

i is an index for the letterset determined in equation 83 (see tables 8–10) utilizing the appropriate zone, and

20 is the number of letters used to subdivide a northing distance of 500,000 m.

As 20 northing area designators cannot cover the entire north–south range of the Moon, they are repeated to fully cover the north–south range in the LTM area.

Table 8. Letterset 1 for the 25-kilometer-grid northing area designators for the Lunar Transverse Mercator range of the Lunar Grid Reference System (LGRS).

[Letters O and I are omitted. *N*25*k*, LGRS 25-kilometer-grid northing area designator]

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
*N*25*k*	A	B	C	D	E	F	G	H	J	K	L	M	N	P	Q	R	S	T	U	V

Table 9. Letterset 2 for the 25-kilometer-grid northing area designators for the Lunar Transverse Mercator range of the Lunar Grid Reference System (LGRS).

[Letters O and I are omitted. *N*25*k*, LGRS 25-kilometer-grid northing area designator]

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
*N*25*k*	F	G	H	J	K	L	M	N	P	Q	R	S	T	U	V	A	B	C	D	E

Table 10. Letterset 3 for the 25-kilometer-grid northing area designators for the Lunar Transverse Mercator range of the Lunar Grid Reference System (LGRS).

[Letters O and I are omitted. *N*25*k*, LGRS 25-kilometer-grid northing area designator]

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
*N*25*k*	L	M	N	P	Q	R	S	T	U	V	A	B	C	D	E	F	G	H	J	K

Easting and Northing Coordinates

The final step to determine a full LGRS coordinate is the calculation of the easting and northing position within the 25-km-grid area, parts IV and V of an LGRS coordinate. This is relatively straightforward using modular arithmetic in the following equations:

$$E = X \,\%\, 25{,}000, \tag{85}$$

and

$$N = Y \,\%\, 25{,}000, \tag{86}$$

where

E is the easting coordinate within the 25-km grid area,
N is the northing coordinate within the 25-km grid area, and
X and Y are input easting and northing coordinates from the LTM system.

Precision and Coordinate Truncation

As mentioned previously, LGRS coordinates can be truncated to represent a variable precision, a property of parts IV and V of an LGRS coordinate. This precision is variable and can be specified as needed. By truncating a coordinate, the shortened value will specify an area of the associated precision (table 11).

Table 11. Precision of the Lunar Grid Reference System (LGRS) 25-kilometer easting and northing coordinates.

[m, meter]

Precision (p), in meters	Number of digits (*l*)
1	5
10	4
100	3
1,000 and 10,000	2
25,000	0

LGRS uses a 25-km grid. The smallest natural precision that can be specified when truncating a coordinate is 1 km; thus, the shortest a coordinate can be is 2 numbers. A simple conversion is to truncate the coordinates following NGA (2014b). l is the LGRS easting and northing coordinate length, and p is the desired precision. If $l = 0$, there are no coordinate digits needed and the coordinate precision is 25 km. Note that l cannot be 1 as 10 km cannot be specified naturally in a 25-km grid. If $l = \{2 - 5\}$, then the easting coordinate can be truncated using the following equation:

$$E = \mathrm{int}\left(E / \mathrm{p}\right) \times \mathrm{p} \tag{87}$$

where

p is the specified precision, and
int denotes an integer conversion.

Similarly, the northing coordinate can be truncated using the following equation:

$$N = \mathrm{int}\left(N / \mathrm{p}\right) \times \mathrm{p}\,. \tag{88}$$

LGRS Grid Conversion Equations for the LTM Portion

The equations presented in this section are used to convert between LGRS and LTM coordinates. The coordinates must be already converted from planetocentric latitude and longitude using the appropriate parameters. See the "LTM System Conversion Equations" section for more information.

LGRS Grid Forward Conversion for the LTM Portion

We refer to the forward conversion from LTM coordinates to LGRS coordinates with the following general mapping statement:

$$LTM:(E,N,Z,H)\overset{LGRS}{\rightarrow}\{lonBand,latBand,E25k,N25k,E,N\}, \tag{89}$$

where

lonBand is the LTM zone, Z,
latBand is the LGRS 8° latitudinal band,
*E*25*k* and *N*25*k* are the easting and northing 25-km-grid area designators, and
E and *N* are easting and northing coordinates within the 25-km grid, truncated to the desired precision.

Tables for looking up area designators are included in the associated software provided in appendixes 1 and 2 and ordered according to tables 6 through 11. Once defined, the first portion of LGRS, the LTM zone, and the latitudinal band are determined. The LTM zone defines the LGRS longitude band and is used as the output term *lonBand* without modification. *latBand* is determined by converting back to latitude and longitude and then entering the latitude into equation 81 to determine the correct 8° band using table 6. The 25-km-grid easting and northing area designators are then determined. For the easting area designator, use equation 82 and table 7. As there is no difference in *E*25*k* in adjacent zones, there is no need to modify this equation. To determine the correct letterset for *N*25*k*, use equation 83 and see tables 8 through 10. Once the northing letterset is determined, the 25-km-grid northing area designator is determined from equation 84. See table 12 to compare the grid area designator layout to LTM zone coordinates.

The remaining easting and northing coordinates within the 25-km grid are then determined using equations 85 and 86. The resulting coordinate can be truncated to the desired precision using equations 87 and 88 and table 11.

LGRS Grid Inverse Conversion for LTM Portion

The inverse conversion uses known LGRS coordinates to reconstruct LTM coordinates. The inverse conversion is an additive process, where each part of the LGRS grid coordinate is summed to reproduce the original LTM coordinate position. We refer to the inverse conversion from LGRS coordinates (in the LTM range) to LTM coordinates with the following general mapping statement:

$$LGRS:(lonBand,latBand,E25k,N25k,E,N)\overset{LTM}{\rightarrow}\{E,N,Z,H\}. \tag{90}$$

The first step in reconstructing the LTM coordinate is to determine the correct zone. As the *lonBand* value is the same as the LTM zone, *Z*, this value is reused without modification. Thus, *lonBand* = *Z*.

The LTM hemisphere is also needed to determine the inverse conversion. The *latBand* can be used to determine if a coordinate is in the lunar northern or southern. This can be determined using a reverse string index search function, which we denote using the following equation:

$$i=index(char), \tag{91}$$

where

i is the index reference of a character input, *char* when applied to a string.

To determine if the hemisphere is northern or southern, we apply the inverse conversion from LGRS to latitude and longitude (eq. 90) and use table 3 in the following equation:

$$\begin{gathered}\text{if } index(latBand)\geq index('\text{N}');\\ H=\text{"N"}\\ \text{else if } index(latBand)<index('\text{N}');\\ H=\text{"S"}.\end{gathered} \tag{92}$$

Thus, if the letter index is greater or equal to the index of band N, the hemisphere is northern (H = "N"); if the letter index is less than the index of band N, the hemisphere is southern (H = "S") (see fig. 15).

Next, convert the 25-km-grid easting and northing area designators back to the position of the 25-km grid. The area designators reference the lower left corner of the 25-km-grid areas; conversion to LTM provides a coordinate of the lower left corner position.

The 25-km-grid easting position is determined using the following equation and table 6:

$$E25k_{\text{num}} = 5 + index\left(E25k\right) \times 25{,}000\,, \tag{93}$$

where

$E25k_{\text{num}}$ is the LTM easting position of the lower left corner of the 25-km grid.

Determining the northing area designator requires the additional step of determining the correct letter positioning as there are three letter sets for the LGRS 25-km-grid northing area designator (tables 8–10, 12). This can be completed using the zone as determined from the *lonBand*. The correct letter index can be determined from equations 83 and 84. See table 12 for the overall layout. Once the index location is known the 25-km-grid northing position is determined using the following equation and tables 8–10:

$$N25k_{\text{num}} = index\left(N25k\right) \times 25{,}000\,, \tag{94}$$

where

$N25k_{\text{num}}$ is the LTM northing position of the lower left corner of the 25-km grid.

Note that $N25k_{\text{num}}$ only provides the relative position of the 25-km-grid area within table 12. This means the $N25k_{\text{num}}$ is within the range of 0 to 500,000 m and needs to be adjusted to its final position. The grid summarized in table 12 is repeated multiple times, every 500,000 m to fully cover the north–south range of the Moon in the LTM portion. To extract the final LTM northing position of the lower left corner of the 25-km-grid area, the northing position of the entire LGRS 25-km letter grouping needs to be determined. We refer to this location as $N2M$ representing the southernmost position of the 25-km letter grouping shown in table 12.

Let φ_{band} be the latitude of the bottom of an LGRS 8° latitudinal band. To extract the position of φ_{band}, the northing position of *latBand* is calculated using

$$\varphi_{\text{band}} = index\left(latBand\right) - 11 \times 8°\,, \tag{95}$$

by indexing table 6, equation 95 determines the inverse of the results of equation 81. Subtracting 11 functions as a shift that supports an equator at 0°; thus, when *latBand* = N, $\varphi_{\text{band}} = 0°$. The latitude φ_{band} of the LGRS global latitudinal grid is converted to northings, N_{band}, using the forward conversion from latitude and longitude to LTM coordinates (see "LTM System Conversion Equations" section and equation 18). The LTM northing of the latitudinal band is the only variable of importance for this calculation. N_{band} is calculated using

$$N_{\text{band}} = \left\lfloor \left(N_{\text{band}} / 25{,}000\right) \right\rfloor \times 25{,}000\,. \tag{96}$$

We note that applying equation 96 is not sufficient to determine the final position of $N25k_{\text{num}}$. $N25k_{\text{num}}$ is the grid position within the LGRS 25-km-grid area, a structure that repeats every 500,000 m, and N_{band} currently has its grid position referenced to the bottom of an 8° grid cell within the designated letterset (table 12).

The final global northing position of the 25-km-grid area (table 12), $N2M$, can be determined by iteration. $N2M$ is set to 0 m and increased by an interval of 500,000 m until the summed position of $N2M$, $N25k_{\text{num}}$, and N is greater than the position of N_{band}. At this location, $N2M$ will have been iterated to specify the correct location. This iteration process can be thought of as stacking multiple occurrences of the 25-km-grid area designator letter grouping—spaced every 500,000 m in northings (as shown in table 12)—on top of each other until the coordinate position is in the correct LTM position. The iteration is implemented with a while loop as follows:

$$\begin{gathered} N2M = 0, \\ \text{while } (N2M + N25k_{\text{num}} + N < N_{\text{band}}), \\ N2M \mathrel{+}= 500{,}000. \end{gathered} \tag{97}$$

Table 12. 25-kilometer-grid easting and northing area designators for zones in the Lunar Transverse Mercator (LTM) portion of the Lunar Grid Reference System (LGRS).

[This letter structure repeats in the northing direction to fully cover the LTM zone. $N25k_{num}$ and $E25k_{num}$, LGRS 25-kilometer-grid northing and easting position]

Letterset:	Set 1										Set 2										Set 3									
Zone:	--	1, 22	4, 25	7, 28	10, 31	13, 34	16, 37	19, 40	43	--	--	2, 23	5, 26	8, 29	11, 32	14, 35	17, 38	20, 41	44	--	--	3, 24	6, 27	9, 30	12, 33	15, 36	18, 39	21, 42	45	--
	$E25k_{num}$																													
$N25k_{num}$	125,000	150,000	175,000	200,000	225,000	250,000	275,000	300,000	325,000	350,000	125,000	150,000	175,000	200,000	225,000	250,000	275,000	300,000	325,000	350,000	125,000	150,000	175,000	200,000	225,000	250,000	275,000	300,000	325,000	350,000
475,000	AV	BV	CV	DV	EV	FV	GV	HV	JV	KV	AE	BE	CE	DE	EE	FE	GE	HE	JE	KE	AK	BK	CK	DK	EK	FK	GK	HK	JK	KK
450,000	AU	BU	CU	DU	EU	FU	GU	HU	JU	KU	AD	BD	CD	DD	ED	FD	GD	HD	JD	KD	AJ	BJ	CJ	DJ	EJ	FJ	GJ	HJ	JJ	KJ
425,000	AT	BT	CT	DT	ET	FT	GT	HT	JT	KT	AC	BC	CC	DC	EC	FC	GC	HC	JC	KC	AH	BH	CH	DH	EH	FH	GH	HH	JH	KH
400,000	AS	BS	CS	DS	ES	FS	GS	HS	JS	KS	AB	BB	CB	DB	EB	FB	GB	HB	JB	KB	AG	BG	CG	DG	EG	FG	GG	HG	JG	KG
375,000	AR	BR	CR	DR	ER	FR	GR	HR	JR	KR	AA	BA	CA	DA	EA	FA	GA	HA	JA	KA	AF	BF	CF	DF	EF	FF	GF	HF	JF	KF
350,000	AQ	BQ	CQ	DQ	EQ	FQ	GQ	HQ	JQ	KQ	AV	BV	CV	DV	EV	FV	GV	HV	JV	KV	AE	BE	CE	DE	EE	FE	GE	HE	JE	KE
325,000	AP	BP	CP	DP	EP	FP	GP	HP	JP	KP	AU	BU	CU	DU	EU	FU	GU	HU	JU	KU	AD	BD	CD	DD	ED	FD	GD	HD	JD	KD
300,000	AN	BN	CN	DN	EN	FN	GN	HN	JN	KN	AT	BT	CT	DT	ET	FT	GT	HT	JT	KT	AC	BC	CC	DC	EC	FC	GC	HC	JC	KC
275,000	AM	BM	CM	DM	EM	FM	GM	HM	JM	KM	AS	BS	CS	DS	ES	FS	GS	HS	JS	KS	AB	BB	CB	DB	EB	FB	GB	HB	JB	KB
250,000	AL	BL	CL	DL	EL	FL	GL	HL	JL	KL	AR	BR	CR	DR	ER	FR	GR	HR	JR	KR	AA	BA	CA	DA	EA	FA	GA	HA	JA	KA
225,000	AK	BK	CK	DK	EK	FK	GK	HK	JK	KK	AQ	BQ	CQ	DQ	EQ	FQ	GQ	HQ	JQ	KQ	AV	BV	CV	DV	EV	FV	GV	HV	JV	KV
200,000	AJ	BJ	CJ	DJ	EJ	FJ	GJ	HJ	JJ	KJ	AP	BP	CP	DP	EP	FP	GP	HP	JP	KP	AU	BU	CU	DU	EU	FU	GU	HU	JU	KU
175,000	AH	BH	CH	DH	EH	FH	GH	HH	JH	KH	AN	BN	CN	DN	EN	FN	GN	HN	JN	KN	AT	BT	CT	DT	ET	FT	GT	HT	JT	KT
150,000	AG	BG	CG	DG	EG	FG	GG	HG	JG	KG	AM	BM	CM	DM	EM	FM	GM	HM	JM	KM	AS	BS	CS	DS	ES	FS	GS	HS	JS	KS
125,000	AF	BF	CF	DF	EF	FF	GF	HF	JF	KF	AL	BL	CL	DL	EL	FL	GL	HL	JL	KL	AR	BR	CR	DR	ER	FR	GR	HR	JR	KR
100,000	AE	BE	CE	DE	EE	FE	GE	HE	JE	KE	AK	BK	CK	DK	EK	FK	GK	HK	JK	KK	AQ	BQ	CQ	DQ	EQ	FQ	GQ	HQ	JQ	KQ
75,000	AD	BD	CD	DD	ED	FD	GD	HD	JD	KD	AJ	BJ	CJ	DJ	EJ	FJ	GJ	HJ	JJ	KJ	AP	BP	CP	DP	EP	FP	GP	HP	JP	KP
50,000	AC	BC	CC	DC	EC	FC	GC	HC	JC	KC	AH	BH	CH	DH	EH	FH	GH	HH	JH	KH	AN	BN	CN	DN	EN	FN	GN	HN	JN	KN
25,000	AB	BB	CB	DB	EB	FB	GB	HB	JB	KB	AG	BG	CG	DG	EG	FG	GG	HG	JG	KG	AM	BM	CM	DM	EM	FM	GM	HM	JM	KM
00,000	AA	BA	CA	DA	EA	FA	GA	HA	JA	KA	AF	BF	CF	DF	EF	FF	GF	HF	JF	KF	AL	BL	CL	DL	EL	FL	GL	HL	JL	KL

This iteration is the inverse of equations 83 and 84.
The final LTM easting coordinate position can be summed to its final position using the following equation:

$$E = E25k_{\text{num}} + E\,. \tag{98}$$

The final LTM northing coordinate position can be calculated using the following equation

$$N = N2M + N25k_{\text{num}} + N\,. \tag{99}$$

Note that in the sample code provided in appendixes 1 and 2, the northing and easting each need to have five digits; truncated coordinate values should be zero-padded to meet this coordinate length and precision. This length is needed so the LGRS inverse conversion can determine their truncated position.

LGRS Coordinate Structure for the LPS Portion

The LPS portion of LGRS coordinates is very similar to the LTM portion as it is a sequence of numbers and letters that are used to define a polar area of the lunar surface. An example of an LGRS coordinate in the LPS portion is as follows:

AZS1359008480 (LGRS coordinate specifying a grid area of 1 m × 1 m)
(lower left corner at lat −86.38231380366628°, long −6.004331982958013°).

From left to right the coordinate is formatted as

I. A letter that specifies the western or eastern portion of an LPS zone. Letters A and B are used for the south pole and letters Y and Z are used for the north pole.

II. A two-letter string that designates the 25-km-grid area designator. The first letter specifies the easting area; the second letter specifies the northing area designator.

III. A 1- to 5-digit number that represents the easting coordinate to the desired precision. Coordinates are referenced to the lower left corner of the 25-km-grid area.

IV. A 1- to 5-digit number that represents the northing coordinate to the desired precision. Coordinates are referenced to the lower left corner of the 25-km-grid area.

Letters I and O are omitted in LGRS. LGRS coordinates are written with no spaces or delimiters. LGRS and associated coordinates are designed with the capability for reporting a relative precision to specify a specific area of variable size. Reporting a variably sized area is completed by truncating the full LGRS coordinate. These coordinates, which are slightly shorter than LGRS coordinates in the LTM range, are not fixed in width as the coordinate length specifies the precision. Zero padding the front of parts III and IV of the coordinate may be needed to specify the correct precision of a grid coordinate. Additionally, the sample code provided requires a 5-digit value of parts IV and V of and LGRS Coordinate to convert position correctly. Zero-padding should be applied to meet this length requirement.

LGRS Grid Scheme for the LPS Portion

This section outlines the structure of coordinates in LGRS for the range of the LPS system (lat −90° to −80° and lat 80° to 90°). The coordinate structure is different from the coordinate structure for the LTM portion. The LPS system uses a single letter to designate an LPS zone; however, a 25-km grid is also used for both the north and south LPS zones.

LGRS Use of LPS Zones

The LGRS uses the north and south polar zones and map projections from the LPS system. The hemisphere, *H*, is used to distinguish the north- and south-polar LPS zones.

LGRS Additions to LPS Zones

The LGRS divides each polar zone further into an east and west area, or “band,” which is designated using one of the letters A, B, Y, or Z. This is part I of an LGRS coordinate. These four letters refer only to polar regions south of lat −80° and north of lat 80° (figs. 16, 17). Letters C through X are in the LTM portion of LGRS and are not applicable in this section. Zone A is the western hemisphere half of the south pole; Zone B is the eastern hemisphere half of the south pole (fig. 17). Zone Y is the western hemisphere half of the north pole; Zone Z is the eastern hemisphere half of the north pole (fig. 16).

The LGRS band of an LPS coordinate in either the north or south polar region is determined by whether the LPS easting coordinate is less than or greater than the LPS false easting value of 500 km. It is calculated using the following equations:

$$\text{if } E < F_E,\ Z = \text{"A" or "Y"}, \tag{100}$$

and

$$\text{if } E \geq F_E,\ Z = \text{"B" or "Z"}. \tag{101}$$

25-km Grid Areas and Designator Scheme

This section describes the 25-km-grid structure for the north- and south-polar ranges of LGRS. For these polar regions, the 25-km-grid layout differs from the LTM portion, as can be seen through a comparison of table 12 to figures 18 and 19; but remains functionally the same. The LPS portion of LGRS uses the full alphabet (omitting letters I and O) to designate more zones. For this transformation from LPS to polar LGRS, LPS coordinates must be in the range of

196,300 m ≤ E < 803,702 m, and
196,300 m ≤ N < 803,702 m.

If this condition is met, the 25-km grid can be used.

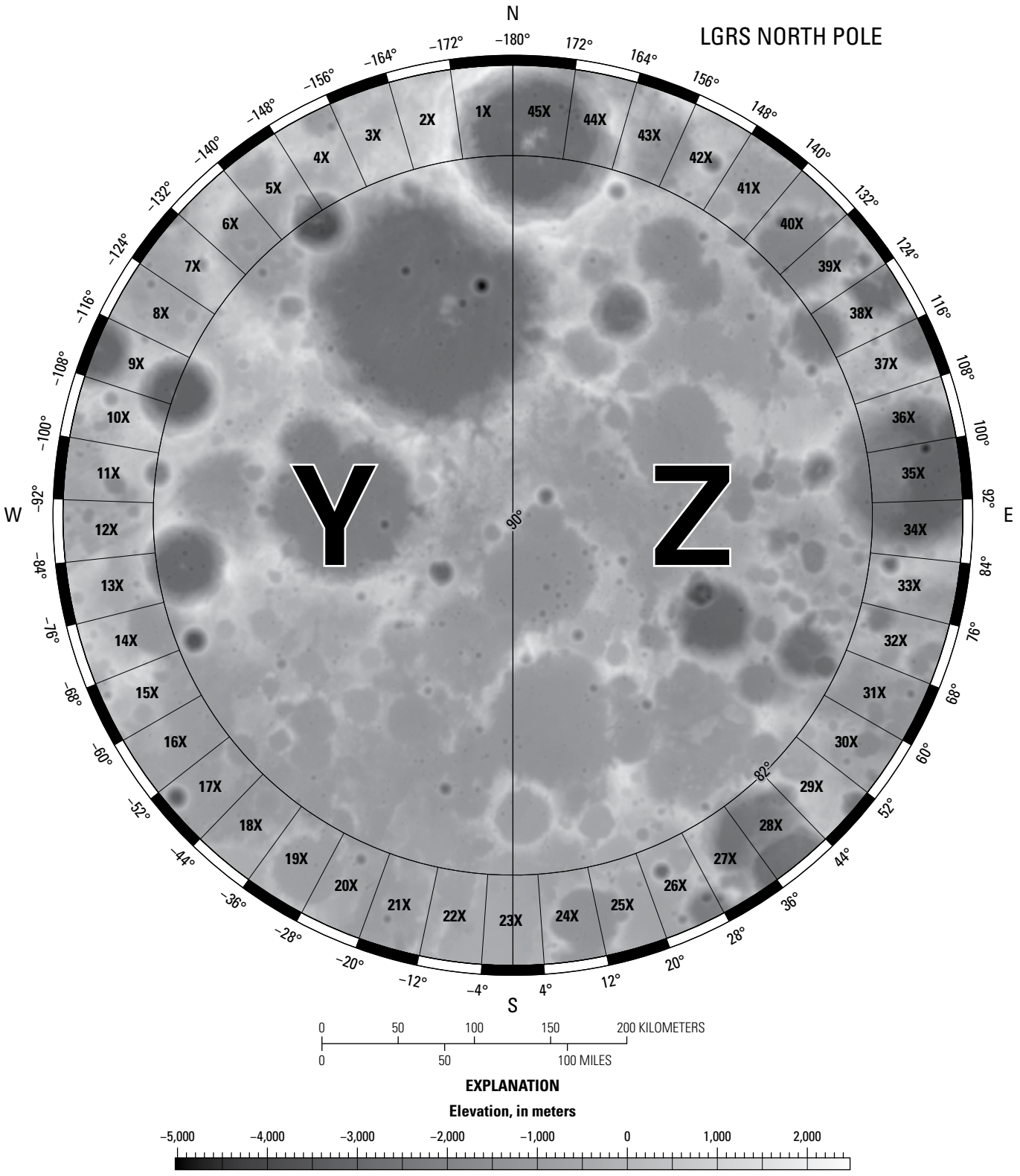


Figure 16. Map showing Lunar Grid Reference System (LGRS) north polar region zones. Designators Y and Z may be used from 80° to 90°. Band-X zones in the extended Lunar Transverse Mercator range are shown; however, polar LGRS coordinates are recommended. Elevation from a Lunar Orbiter Laser Altimeter (LOLA) 118-meter (m)-resolution digital elevation model (LOLA Science Team, 2021).

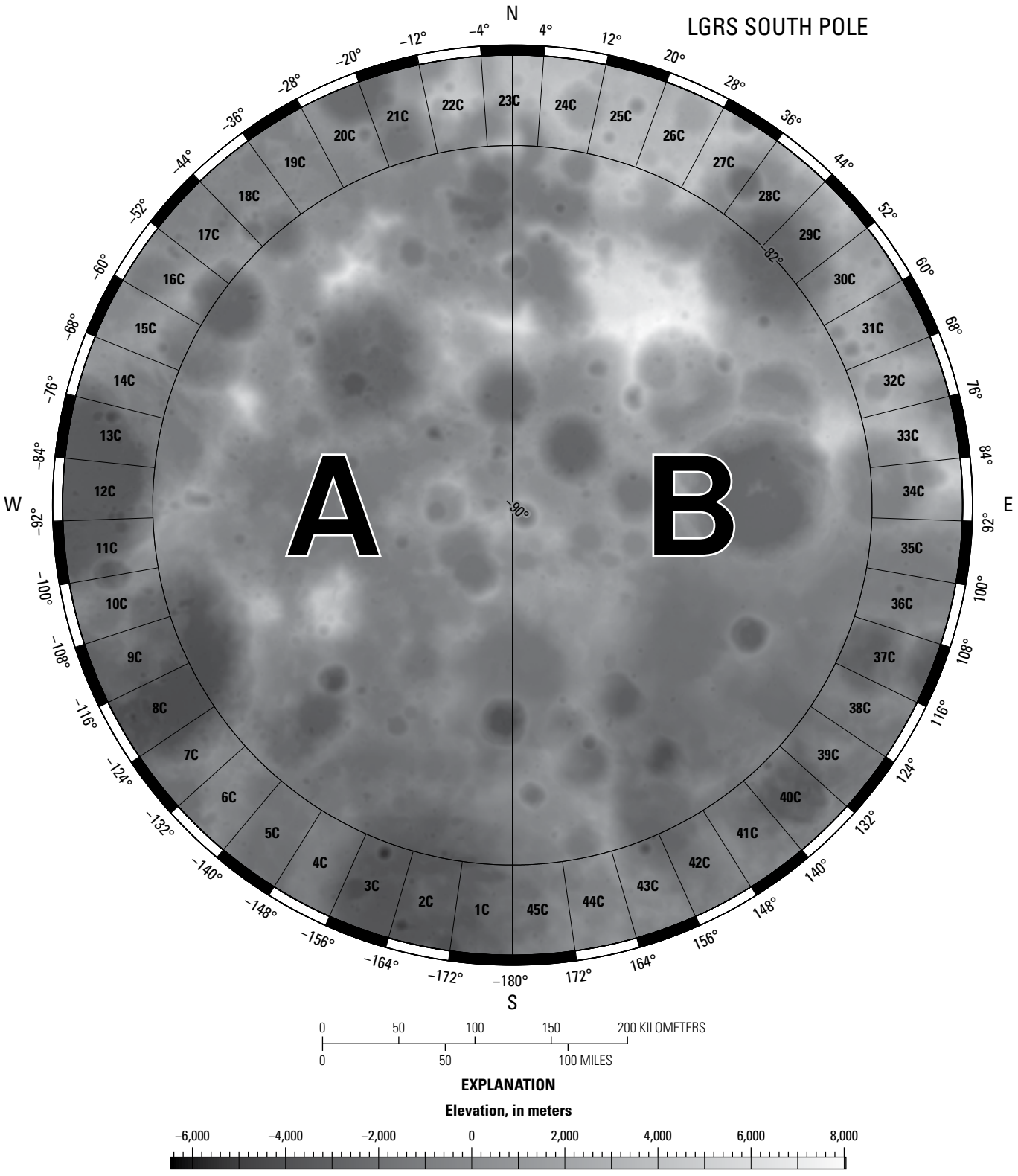


Figure 17. Map showing Lunar Grid Reference System (LGRS) south polar region zones. Designators A and B may be used from –90° to –80°. Band-C zones in the extended Lunar Transverse Mercator range are shown; however, polar LGRS coordinates are recommended. Elevation from a Lunar Orbiter Laser Altimeter (LOLA) 118-meter (m)-resolution digital elevation model (LOLA Science Team, 2021).

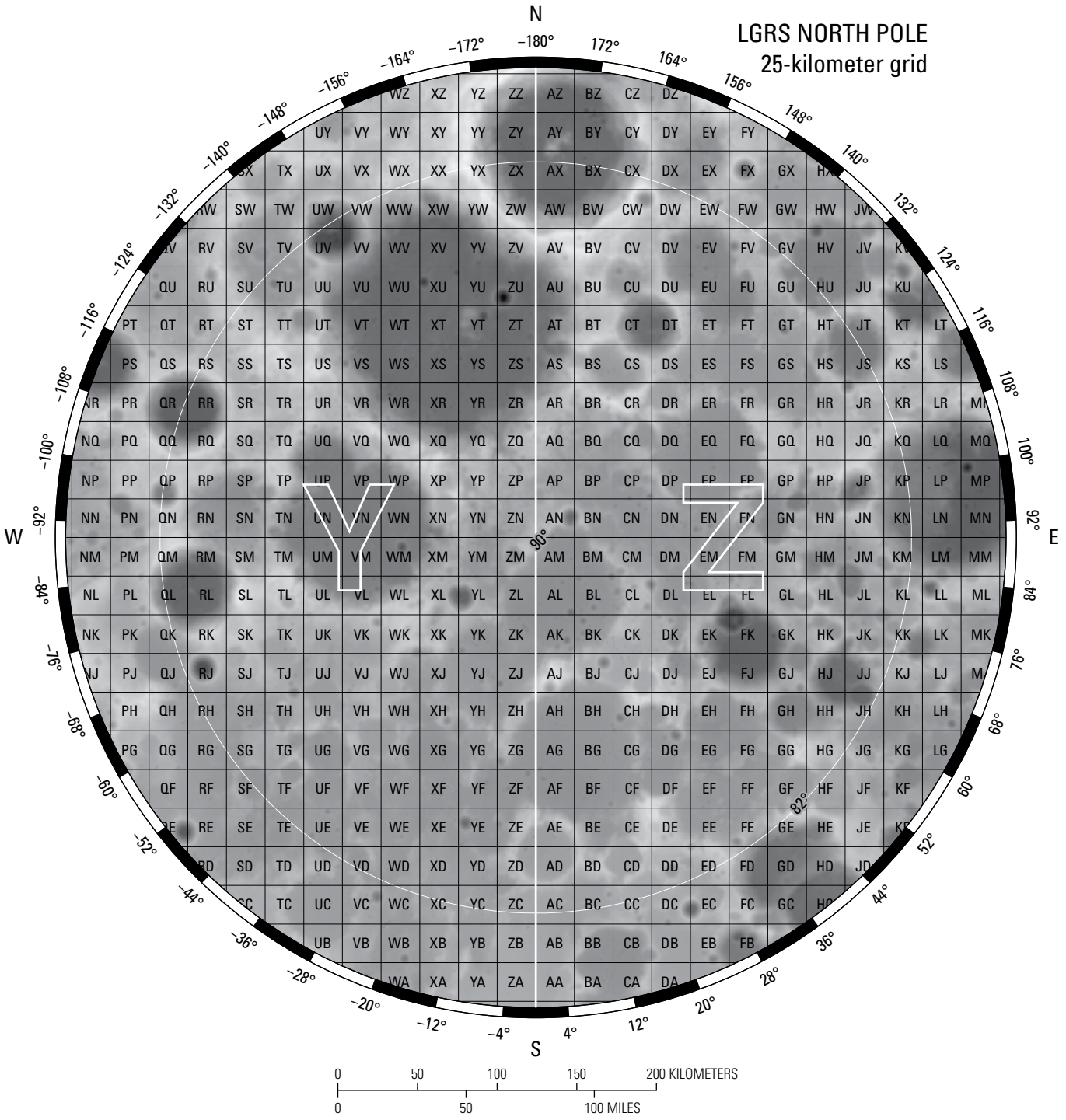


Figure 18. Map showing north polar Lunar Grid Reference System (LGRS) global and 25-kilometer-grid areas and labels. Note that band-X zones in the extended Lunar Transverse Mercator range can be used between 80° and 82°; however, polar LGRS coordinates are recommended.

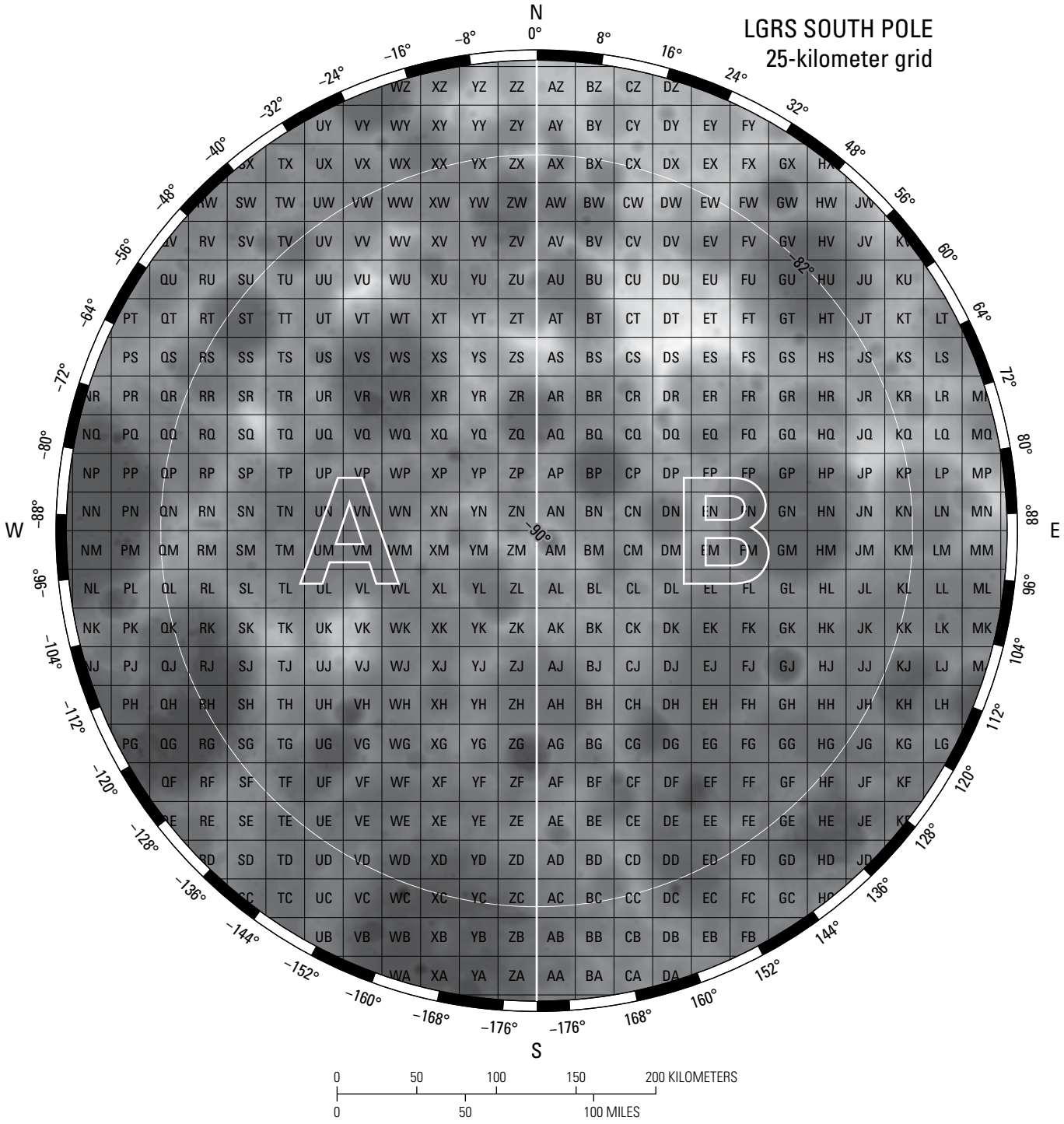


Figure 19. Map showing south polar Lunar Grid Reference System (LGRS) global and 25-kilometer-grid areas and labels. Note that band-C zones in the extended Lunar Transverse Mercator range can be used between –82° and –80°; however, polar LGRS coordinates are recommended.

The 25-km-grid areas, part 2 of the LGRS polar coordinate, are referenced with a two-letter string, where each letter is a 25-km-grid easting or northing area designator. For zones in the eastern half of the polar projection area, the easting area designators use letters A to N (omitting letter I), beginning from the 0° and 180° meridian. For zones in the western band of the polar projection area the easting area designators use letters Z to M (omitting letter O) (figs. 18, 19). Letters M and N are used twice in each LGRS band; although a 25-km grid fits most of the LPS range well, given NASA requirements of condensing coordinates, a 3-km gap is present in the eastern and western margins of the grid area if letters M and N are not reused (figs. 18, 19). See tables 13 and 14 for a full list of easting area designators.

To determine the 25-km-grid easting area designator, we use a table and function indexing methodology similar to those used for the LTM portion. As the easting area designator differs between each band of the projection area, we use two unique letter sets (tables 13, 14). Equations 100 and 101 are used to determine the letterset:

Table 13. 25-kilometer-grid easting area designators for bands A and Y in the Lunar Polar Stereographic portion of the Lunar Grid Reference System (LGRS).

[*E*25*k*, LGRS 25-kilometer-grid easting area designator]

i	12	13	14	15	16	17	18	19	20	21	22	23	24
*E*25*k*	M	N	P	Q	R	S	T	U	V	W	X	Y	Z

Table 14. 25-kilometer-grid easting area designators for bands B and Z in the Lunar Polar Stereographic portion of the Lunar Grid Reference System (LGRS).

[*E*25*k*, LGRS 25-kilometer-grid easting area designator]

i	1	2	3	4	5	6	7	8	9	10	11	12	13
*E*25*k*	A	B	C	D	E	F	G	H	J	K	L	M	N

$$\text{if } E < F_E,\ lonBand = \text{"A" if south or "Y" if north}, \tag{100}$$

and

$$\text{if } E \geq F_E,\ lonBand = \text{"B" if south or "Z" if north}. \tag{101}$$

If the easting is less than the LPS false easting, then table 13 is used; if the easting is greater than the LPS false easting, then table 14 is used.

To correctly index the easting area designators, we remove the false easting from the coordinates to make the values relative to the central, north–south meridian, and divide by the grid dimensions to determine the proper index. For bands B or Z, use table 14 and the following index function:

$$i = \left\lfloor \left(E - F_E\right) / 25{,}000 \right\rfloor, \tag{102}$$

where

i is an index for tables 14, and
$\lfloor \cdot \rfloor$ represents the floor

For bands A and Y, use the following index function:

$$i = 24 - \left\lfloor \left(\left|E\right| - F_E\right) / 25{,}000 \right\rfloor, \tag{103}$$

where

i is an index for table 13,
24 is the total number of designators in tables 13 and 14,
$\lfloor \cdot \rfloor$ represents the floor, and
$\left| \cdot \right|$ is the absolute value of a number.

In summary, the easting area designators increment from letter A in bands B and Z in the eastern half of the polar projection area and decrement from letter Z in areas A and Y in the western half of the polar projection area. This was a design choice of LGRS, and a property of MGRS (figs. 18, 19) (NGA, 2014b). This designation was made (1) to impose directionality on the map where A and Y are west, zones B and Z are east, and longitudes 0° and 180° are north-south, and (2) so that when navigating under field conditions, the change from Z to A or vice versa will indicate to a map user that they have switched LGRS areas (figs. 16, 17).

The 25-km-grid northing area designator uses letters A to Z (omitting letters I and O) from grid south to grid north of each polar projection area. We assign values from the middle of the alphabet, M, N, along the longitude −90° and 90° west–east meridian (figs. 18, 19). We note, similarly to the easting area designator letterset, not enough letters are present to fully cover the entire polar region. As such, a 3-km gap exists at grid north and grid south. To supplement these letters we use the minus sign (−) in the south and the plus sign (+) in the north. See tables 15 and 16 for a full list of the northing area designators.

Table 15. 25-kilometer-grid northing area designators for all polar zones in the Lunar Grid Reference System (LGRS).

[*N*25*k*, LGRS 25-kilometer-grid northing area designator]

i	0	1	2	3	4	5	6	7	8	9	10	11	12
*N*25*k*	−	A	B	C	D	E	F	G	H	J	K	L	M

Table 16. 25-kilometer-grid northing area designators for all polar zones in the Lunar Grid Reference System (LGRS).

[*N*25*k*, LGRS 25-kilometer-grid northing area designator]

i	13	14	15	16	17	18	19	20	21	22	23	24	25
*N*25*k*	N	P	Q	R	S	T	U	V	W	X	Y	Z	+

To determine the correct northing grid area designator, use the following index function:

$$i = \left\lfloor \left(N - F_N \right) / 25{,}000 \right\rfloor + 13, \tag{104}$$

where

i is an index for tables 15 and 16, and
13 is half the number of the 25-km-grid area designators and shifts the reference to the center.

If $\left(N - F_N \right)$ is positive, table 16 is referenced; if $\left(N - F_N \right)$ is negative, the index is still positive, and table 15 is used. There is no need for a conditional use case statement for table 15 and 16.

Easting and Northing Coordinates

The final step to determine a full LGRS coordinate is the calculation of the easting and northing position within the 25-km grid, parts III and IV of an LGRS polar coordinate. At this point the 25-km-grid area has been determined and the easting and northing positions within the 25-km-grid area need to be determined. This is relatively straight forward and utilizes modular arithmetic similar to equations 85 and 86. For the LPS portion of LGRS, use the following equations to determine the easting position.
For bands A and Y,

$$E = 25{,}000 - \left(\left| X \right| \ \% \ 25{,}000 \right), \tag{105}$$

and for bands B and Z,

$$E = X \ \% \ 25{,}000, \tag{106}$$

where

X is the LPS coordinate with the false easting removed.

To determine the northing position, use the following equation:

$$N = Y \ \% \ 25{,}000\,, \tag{107}$$

where

Y is the LPS coordinate with the false northing removed.

Precision and Coordinate Truncation

Similar to the LTM portion, LGRS coordinates for the LPS portion can be truncated to a desired specification. See equations 87 and 88 and table 11 for more information.

LGRS Grid Conversion Equations for the LPS Portion

The equations presented in this section are used to convert between LGRS and LPS coordinates. The coordinates must be already converted from planetocentric latitude and longitude using the appropriate parameters. See the "LPS System Conversion Equations" section for more information.

LGRS Grid Forward Conversion for the LPS Portion

We refer to the forward conversion from LPS coordinates to LGRS coordinates with the following general mapping statement:

$$LPS:\left(E,N,H\right) \overset{LGRS\_polar}{\rightarrow} \left\{lonBand, E25k, N25k, E, N\right\}, \tag{108}$$

where

$lonBand$ is the letter designating the western or eastern band, half of an LPS zone,
$E25k$ and $N25k$ are the 25-km-grid easting and northing area designators, and
E and N are easting and northing coordinates within the 25-km-grid area, truncated to the desired precision.

To complete the conversion from LPS to LGRS coordinates, the LGRS band is determined using equations 100 and 101. Next, the polar false origins are removed from the LPS easting and northing positions using the following equations:

$$X = E - F_E\,, \tag{109}$$

and

$$Y = N - F_N\,. \tag{110}$$

This references the coordinates to the center of the LPS zone.

The 25-km-grid easting and northing area designators are then determined. For the easting area designator, if the coordinate is in zone A or Y, use equation 103 and table 13. If the coordinate is in zone B or Z, use equation 102 and table 14. For the northing area designator, use equation 104 and tables 15 and 16.

Once the area designators have been determined, the remaining easting and northing coordinates are referenced to the lower left corner of the 25-km grid. Eastings and northings are determined using equations 105 through 107. The easting coordinate needs special processing depending on the coordinate location in $lonBand$ as our methods have removed the false easting from the coordinate. The coordinate then can be truncated to the desired precision using equations 87 and 88.

LGRS Grid Inverse Conversion for the LPS Portion

The inverse conversion of LGRS coordinates to LPS is similar to the inverse conversion of LGRS coordinates to LTM. The inverse conversion is an additive process where each part of the LGRS grid coordinate is summed to reproduce the original LPS coordinate position. We refer to the inverse conversion from LGRS coordinates (in the LPS portion) to LPS coordinates with the following general mapping statement:

$$LGRS\left(lonBand, E25k, N25k, E, N\right) \overset{LPS}{\rightarrow} \left\{E, N, H\right\}. \tag{111}$$

The first step in reconstructing the LPS coordinate is to determine the correct hemispheric location of the coordinate. If *lonBand* is A or B then the hemisphere, *H*, is south, *H* = "S"; if *lonBand* is Y or Z, *H* is north, *H* = "N". This value distinguishes the LPS map projection and the sign of the latitude during the inversion.

The inversion uses the same reverse string index function used for the LTM portion (eq. 91) to determine the indices of the coordinate designators.

The next step is to convert 25-km grid easting and northing area designators back to the numeric positions of the lower left corner of the 25-km-grid area. As the coordinate is referenced to the center of the LPS projection area, two lettersets were used for the easting coordinate position (see tables 13, 14). If the *lonBand* is A or Y, the location is in the western half of the LPS zone and the easting grid position, $E25k_{\text{num}}$, is determined from the following index equation and table 13:

$$E25k_{\text{num}} = -1 \times 25{,}000 \times \left(24 - index\left(E25k\right) + 1\right), \tag{112}$$

where

24 is the number of zone designators in tables 13 and 14.

The index value in equation 112 is multiplied by 25,000 to reposition the grid to its LPS coordinate. The values are also multiplied by −1 to ensure the correct positioning is in LGRS *lonBand* A or Y. As this effectively acts as a rotation, the position is switched from the lower left corner to the right. To correct for this, 1 is added to the reference index. If the *lonBand* is B or Z, the location is in the western band of the LPS zone and the easting grid position is determined using the following index equation and table 14:

$$E25k_{\text{num}} = 25{,}000 \times index\left(E25k\right). \tag{113}$$

The northing grid position, $N25k_{\text{num}}$, is determined using the following index equation and tables 15 and 16:

$$N25k_{\text{num}} = 25{,}000 \times \left(index\left(N25k\right) - 13\right), \tag{114}$$

where

13 is half the number of 25-km-grid northing area designators.

The variables $E25k_{\text{num}}$ and $N25k_{\text{num}}$ only give the position of the lower left corner of the 25-km-grid area. The final LPS coordinate easting can be summed to its final position by re-adding the false easting:

$$E = E25k_{num} + E + F_E, \tag{115}$$

where

$E25k_{\text{num}}$ is the easting position of the lower left corner of the 25km grid area, and

E is the easting position within the 25-km-grid area.

The final LPS coordinate northing can be summed to its final position by re-adding the false northing:

$$N = N25k_{num} + N + F_N, \tag{116}$$

where

$N25k_{\text{num}}$ is the northing position of the lower left corner of the 25km grid area, and

N is the northing position within the 25-km-grid area.

LGRS Sample Conversion and Grid Generation Program

Preliminary software to complete the forward and inverse LGRS conversions is provided in appendix 1. Additional equations to create shapefiles of the LGRS grids are provided in appendix 2.

Artemis Condensed Coordinates (ACC)

The ACC grid format is both an additional grid nested within the LGRS 25-km grid system and a way to shorten the LGRS coordinates to the desired 6-character limit. The 6-character limit, outlined by requirement 2 from NASA's Flight Operations Directorate to the AGDT (McClernan and others, 2023), is intended to align with human abilities to recall information and with the coordinate structure used in the Apollo map books (NASA, 1969; Jones, 2017). Requirement 3 from NASA's Flight Operations Directorate to the AGDT (McClernan and others, 2023) specifies 10-meter precision, which is based on community inputs for real-time precision requirements. We consider ACC as an additional grid and coordinate system built upon LGRS. Thus, LGRS with ACC meets the requirements for lunar surface navigation with a grid system. ACC is not a standalone grid system and requires LGRS to reference ACC globally. This section describes the ACC structure and the conversions between LGRS and LGRS in ACC format.

ACC Grid Format and Coordinate Structure

The coordinate structure of LGRS is composed of a series of letters and numbers which include (1) a global area grid reference, (2) a 25-km-grid area designator, and (3) grid coordinates providing the numerical position relative to the lower left corner of the 25-km-grid area expressed to a desired precision. LGRS coordinates are defined with alphabetic and numeric coordinates of variable length. They are written with no spaces, dashes, or delimiters. LGRS and associated coordinates are designed with the capability for reporting a relative precision to specify a specific area of variable size. Reporting a variable-sized area is completed by truncating the full LGRS coordinate. An example of an LGRS coordinate in the LPS range is as follows:

ZAH**2094****0621**7 (LGRS coordinate specifying a grid area of 1 m × 1 m)
(lower left corner at lat 86.0000°, long 10.0000°)

Note that the first four digits of the easting and the northing parts of the coordinate (**2094** and **0621**, respectively; also in bold) represent 10-m precision.

The ACC format is used to shorten 13- to 15-character LGRS coordinates to 6 characters. To complete this step, an assumption is made that the 25-km-grid area that a point is located in is known. Given this assumption, the LGRS zone, latitudinal band, and 25-km-grid area designators do not need to be reported. This leaves the 10 characters of the easting and northing coordinates that reference the lower left-hand corner of the 25-km grid. The LGRS coordinate specifying a grid area of 1 m × 1 m can be truncated to achieve 10-m precision. This leaves eight remaining characters specifying easting and northing coordinates.

To condense the LGRS coordinates further to ACC, a 1-km grid labeled alphabetically is dimensioned within the 25-km grid. As the LGRS uses a 25-km grid for both the LTM and LPS portions, a 1-km grid will fit universally within all 25-km LGRS grid squares. Additionally, a 1-km grid is an ideal size as a 10-km grid does not neatly fit within the 25-km grid. The ACC format also utilizes a 100-m and 10-m grid labeled numerically within the 25-km LGRS grid structure.

Figure 20 shows an example of an LGRS coordinate in ACC format. The last six characters are reported during field operations and are referred to as ACC. It is assumed the landing location can be used to determine the LGRS zone and the 25-km-grid area; thus, it is omitted in field reporting. The *X* coordinate and *Y* coordinate letters report the 1-km grid and are referenced to the lower left corner of the 25-km-grid area. Hundreds and tens of meters within the 1-km grid are reported with numerical values. The highest precision ACC can report is a 10 m × 10 m area within the known 25-km-grid area. Single meters are omitted during field reporting but can be reported if a precision of single meters is needed. See figure 20 for a breakdown of an LGRS Coordinate Structure in ACC format. The following sections describe the derivation of this coordinate structure. We note the ACC grid labeling structure can be used for any arbitrary 25 km × 25 km lunar map area without repeating grid labels; however, the 25-km-grid areas need to be referenced.

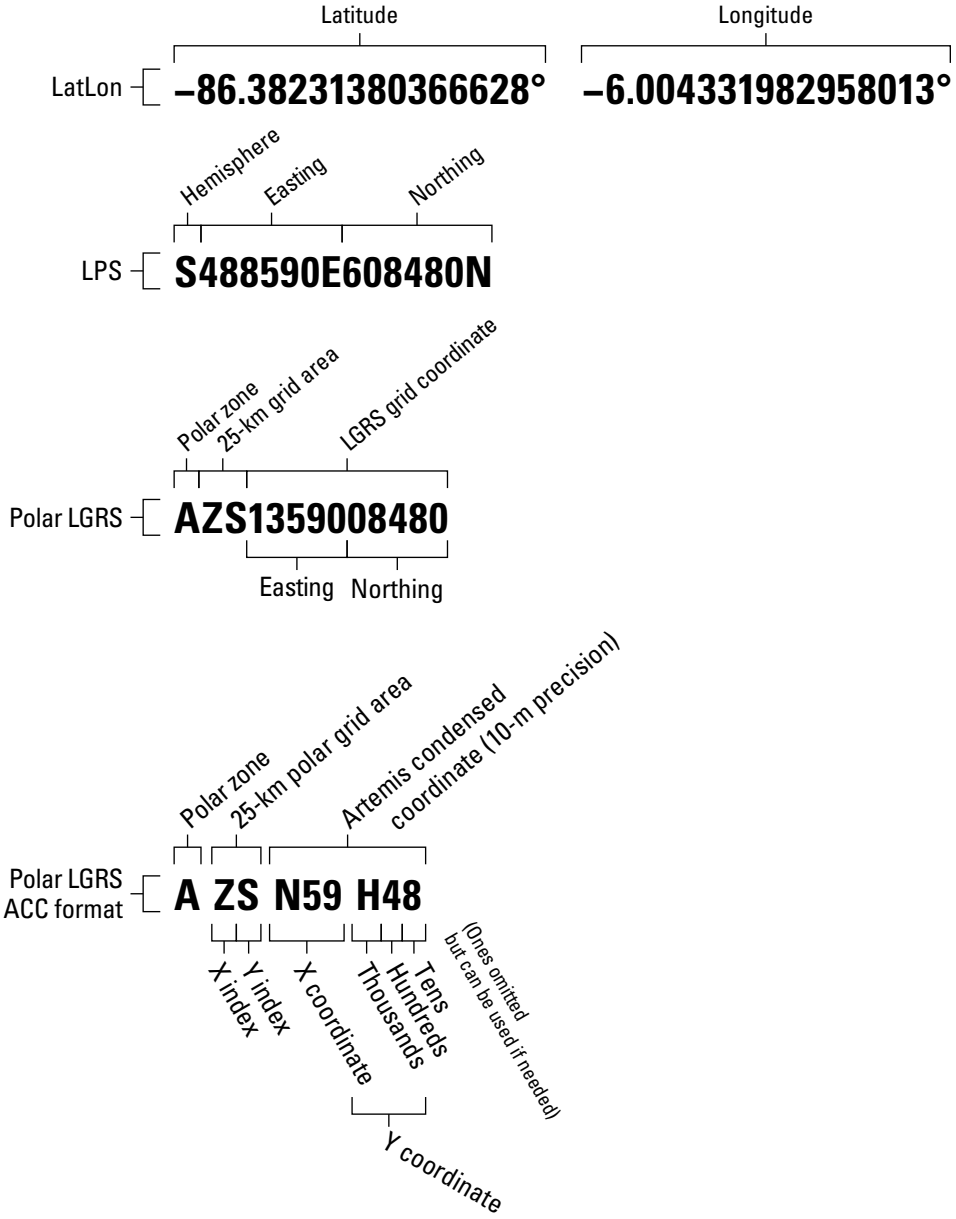


Figure 20. Example format of Lunar latitude and longitude (LatLon), Lunar Polar Stereographic (LPS), polar Lunar Reference Grid System (LGRS), and polar LGRS Artemis Condensed Coordinate (ACC) structures for lunar latitude –86.3823° and longitude –6.0043°. Note that only the last six characters of the LGRS in ACC format are reported in field conditions. m, meter; km, kilometer.

LGRS Grid Coordinate Designations

The 25-km LGRS grids are used as the basis of ACC as grids are subset within the 25-km range. As LGRS defines a grid of rectangular coordinates it is relatively easy to subset additional grids within the 25-km grid. To complete this transformation, coordinates must already be converted from LTM or LPS to LGRS using equation 89 (LTM) or equation 108 (LPS). An example of a 25-km-grid area designator specifying a grid area of 25 km × 25 km in the western south polar zone of the LGRS is “AZS”. To meet the coordinate length requirements of the ACC format, the zone and 25-km-grid area are not included in ACC. Switching grid areas is expected to happen infrequently during field operations, and if a switch occurs the 1-km grid is designed in such a way that switching grid areas can be observed without having to directly state the 25-km-grid area. Switching grid areas can be observed by a shift in the 1-km ACC grid lettering structure. Reporting the LGRS zone and 25-km-grid area is not forbidden in ACC; however, ACC is designed assuming the LGRS zone and 25-km-grid area are common knowledge. The information may be provided at any point to clarify location.

Table 17. 1-kilometer-grid area designators for Artemis Condensed Coordinates (ACC) grids nested within each Lunar Grid Reference System 25-kilometer-grid area.

[Letters and their assigned distance are in northing and easting. km, kilometer; m, meter]

Index (*i*)	1-km-grid area designator	Position (m)	Index (*i*)	1-km-grid area designator	Position (m)
0	-	0,000	13	N	13,000
1	A	1,000	14	P	14,000
2	B	2,000	15	Q	15,000
3	C	3,000	16	R	16,000
4	D	4,000	17	S	17,000
5	E	5,000	18	T	18,000
6	F	6,000	19	U	19,000
7	G	7,000	20	V	20,000
8	H	8,000	21	W	21,000
9	J	9,000	22	X	22,000
10	K	10,000	23	Y	23,000
11	L	11,000	24	Z	24,000
12	M	12,000			

ACC 1-km Grid Coordinate Designations

The first character of both the easting and northing parts of a coordinate in ACC format represent the 1-km grid. To represent the kilometer position within the 25-km grid, letters of the alphabet from A to Z are used, omitting I and O. A hyphen (-) is used to supplement the letters for a total of 25 designators. Thus the hyphen corresponds to 0 and the letter A corresponds to 1 km (table 17). In the future another character may be selected in place of the hyphen based on results of field testing.

To complete this coordinate conversion, the LGRS easting and northing coordinates must be greater than or equal to 0 and less than 25,000 m. At this point, in a similar fashion to the transformation of the 25-km grid, the 1-km-grid area designators are determined using the following index equations. For the 1-km-grid easting area designator,

$$i = \lfloor E/1{,}000 \rfloor, \tag{117}$$

and for the 1-km-grid northing area designator,

$$i = \lfloor N/1{,}000 \rfloor, \tag{118}$$

where

i is an index for table 17.

Like LGRS, the coordinates reference the lower left corner of the 25-km-grid area; thus, each letter represents a 1-km rectangular distance measured from the lower left corner. In field conditions and reporting, the eastings are always reported before the northings. If the 25-km-grid area is known, a coordinate for a 1-km grid can be truncated as shown in table 18.

Table 18. Full and truncated coordinates specifying grids by area in the Lunar Grid Reference System (LGRS).

[--, not applicable; km, kilometer; m, meter]

LGRS coordinate	Truncated coordinate	Grid area
A	--	(Global zone)
AZS	--	25 km × 25 km
AZSNH	NH	1 km × 1 km
AZSN5H4	N5H4	100 m × 100 m
AZSN59H48	N59H48	10 m × 10 m

ACC 100-m and 10-m Grid Coordinate Designations

The 100-m and 10-m grids are referenced using numbers from 0 to 9 for both the easting and northing values. The values of the 100-m and 10-m grid are reported relative to the lower left corner of each 1-km-grid area. The respective values can be used to measure a distance as these values are equivalent to the grid size; for example, a 2 in the 100-m grid represents 200 m and a 5 in the 10-m grid represents 50 m. Equations 87 and 88 are used to determine these truncated coordinate values. Once the precision of a coordinate is determined, the grid values are reported immediately after the 1-km-grid area designator with no spaces or delimiters. Easting always precedes northing.

If only 1-km precision is needed, then the easting and northing values can be omitted (see fig. 20). When LGRS is truncated to 10-m precision, and the LGRS global and 25-km-grid area designators are omitted, the relative position within an LGRS 25-km-grid area can be reported using the required six digits. This is the general formation of LGRS in ACC format. Example plots are provided in figures 21 through 23.

Figure 21 shows the 25-km LGRS grid of the lunar south pole and the Amundsen Rim candidate landing site (blue box; LGRS 25-km-grid area BGQ). Figure 22 shows a 1-km truncated ACC grid of the Amundsen Rim candidate landing site. Note that as depicted, the LGRS coordinates have been dropped and the area designators are truncated to 1-km precision. The teal box (LGRS ACC 1-km-grid area BGQKN) is the 1-km-grid area shown in part *A* of figure 23. Figure 23 shows higher resolutions of ACC, including the 100-m and 10-m grids (parts *A* and *B*, respectively). As ACC coordinates are relative, any 25 km × 25 km area may be uniquely reported using ACC format.

LGRS in ACC format can indicate to a lunar-surface navigator if they switch 25-km-grid areas. This is simply noticed when the six characters switch from the letter Z to "-" or from "-" to Z. This change only happens when the 25-km-grid area is changed. This will be noticed in the northing or easting values and, depending on the direction of travel, used to determine the new 25-km-grid area.

ACC Conversion Equations for LGRS 25-km-Grid Areas

This section describes the conversion process between LGRS coordinates and LGRS coordinates in ACC format. Some precision is lost because of the truncation of coordinates in the conversion process. We have included the capability to estimate this original position and describe the process below.

ACC Forward Conversion for LGRS 25-km-Grid Areas

We refer to the forward conversion from full LGRS coordinates to LGRS coordinates in ACC format with the following general mapping statement:

$$LGRS:\left(lonBand, latBand, E25k, N25k, E, N\right)\overset{ACC}{\rightarrow}\left\{E, N\right\}. \tag{119}$$

The easting and northing coordinates must be formatted in LGRS. To convert LTM coordinates use equation 89 and to convert LPS coordinates use equation 108.

This conversion only applies to the easting and northing coordinates, thus, only LGRS E and N values are needed. First, ensure the coordinate easting and northing values are between 0 and 25,000 m. Next, determine the 1-km-grid easting area designator, $E1k$, using equation 117 and the 1-km-grid northing area designator, $N1k$, using equation 118; both equations reference table 17. The coordinates are then truncated to the desired precision (1 km, 100 m, or 10 m) using equations 87 and 88. To meet the six-character limit, LGRS coordinates in ACC format are truncated to the nearest 10 m.

ACC Inverse Conversion for LGRS 25-km-Grid Areas

We refer to the inverse conversion from LGRS coordinates in ACC format to full LGRS coordinates with the following general mapping statement:

$$ACC:\left(E, N\right)\overset{LGRS}{\rightarrow}\left\{E, N\right\}. \tag{120}$$

Use the index function, equation 91, to identify the position of any alphabetic coordinate values contained in a LGRS coordinate in ACC format.

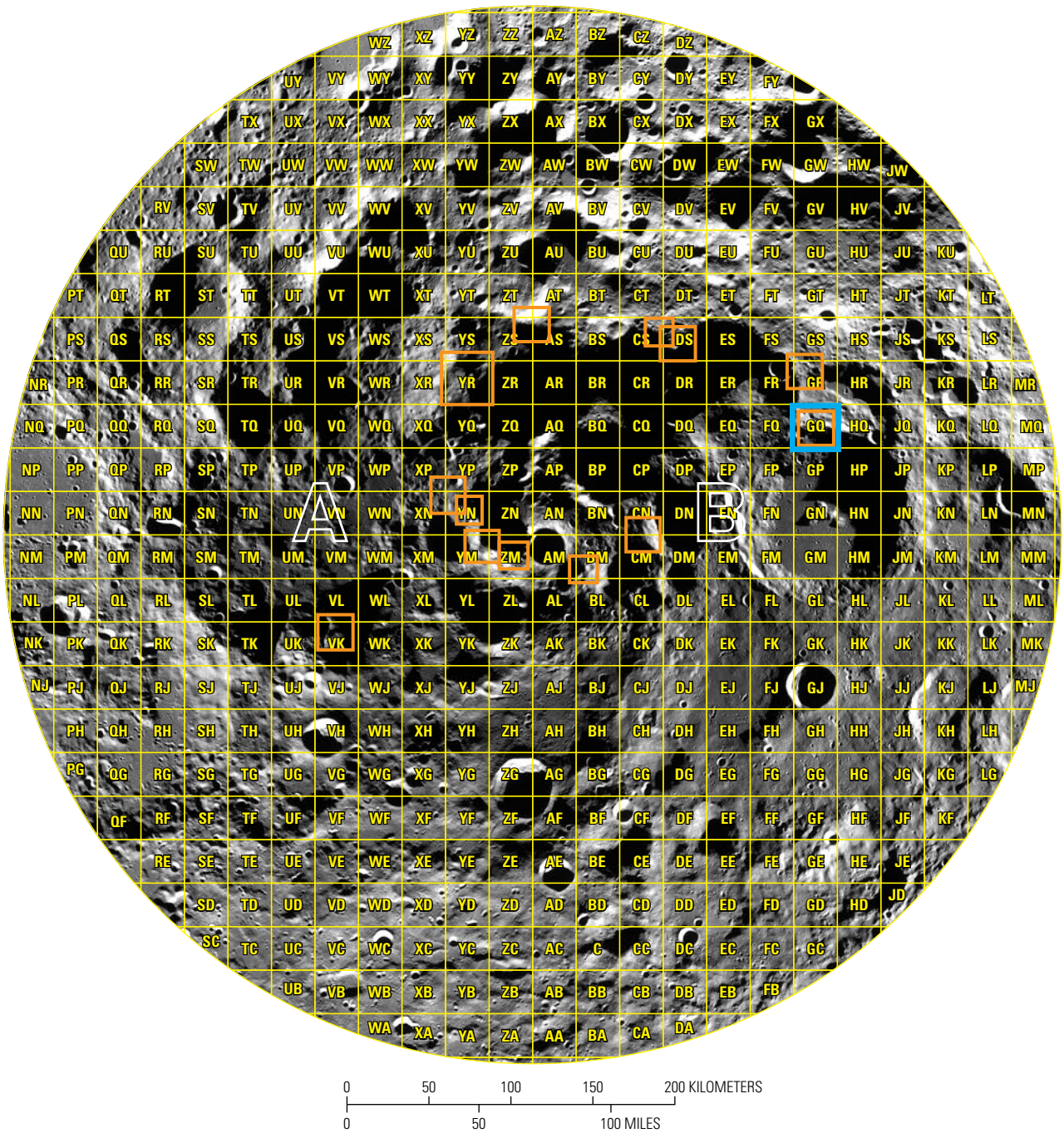


Figure 21. Map of the lunar south pole showing Lunar Grid Reference System (LGRS) global zones with labeled 25-kilometer (km) grid. Orange boxes are Artemis III candidate landing regions. Blue box is the 25-km-grid area containing the Amundsen Rim candidate landing site and area of figure 22. Note all LGRS coordinates are in Artemis Condensed Coordinates (ACC) format. Base from Lunar Reconnaissance Orbiter Camera (LROC) Team (2013).

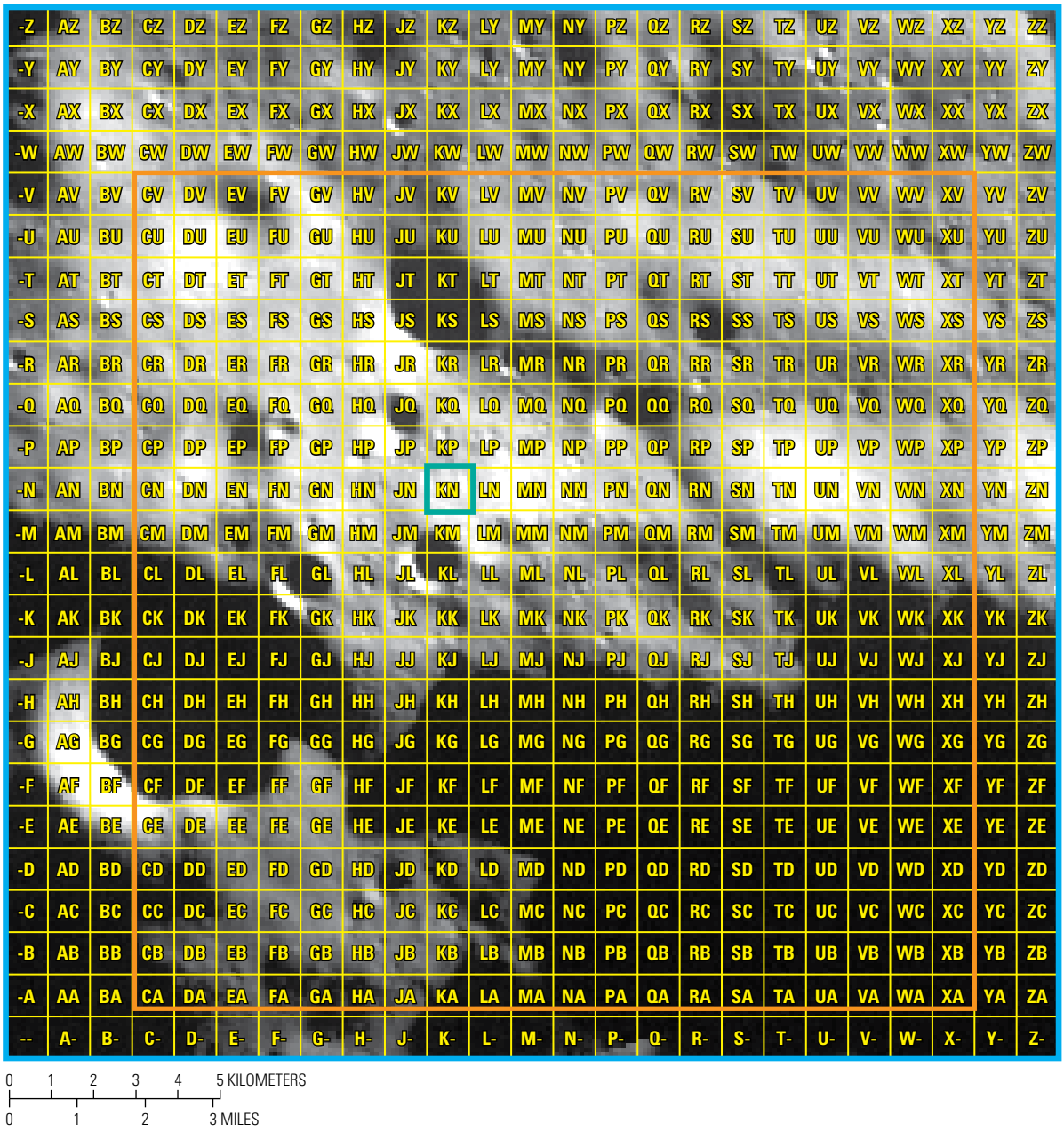


Figure 22. Map of the lunar south pole showing Lunar Grid Reference System (LGRS) 25-kilometer (km)-grid area BGQ (blue outline) with labeled 1-km grid (derived from LGRS 25-km grid shown in figure 21). Orange box is the Artemis III Amundsen Rim candidate landing site. 1-km-grid area BGQKN (teal box) is area of figure 22. Note all LGRS coordinates are in Artemis Condensed Coordinates (ACC) format. Base from Lunar Reconnaissance Orbiter Camera (LROC) Team (2013).

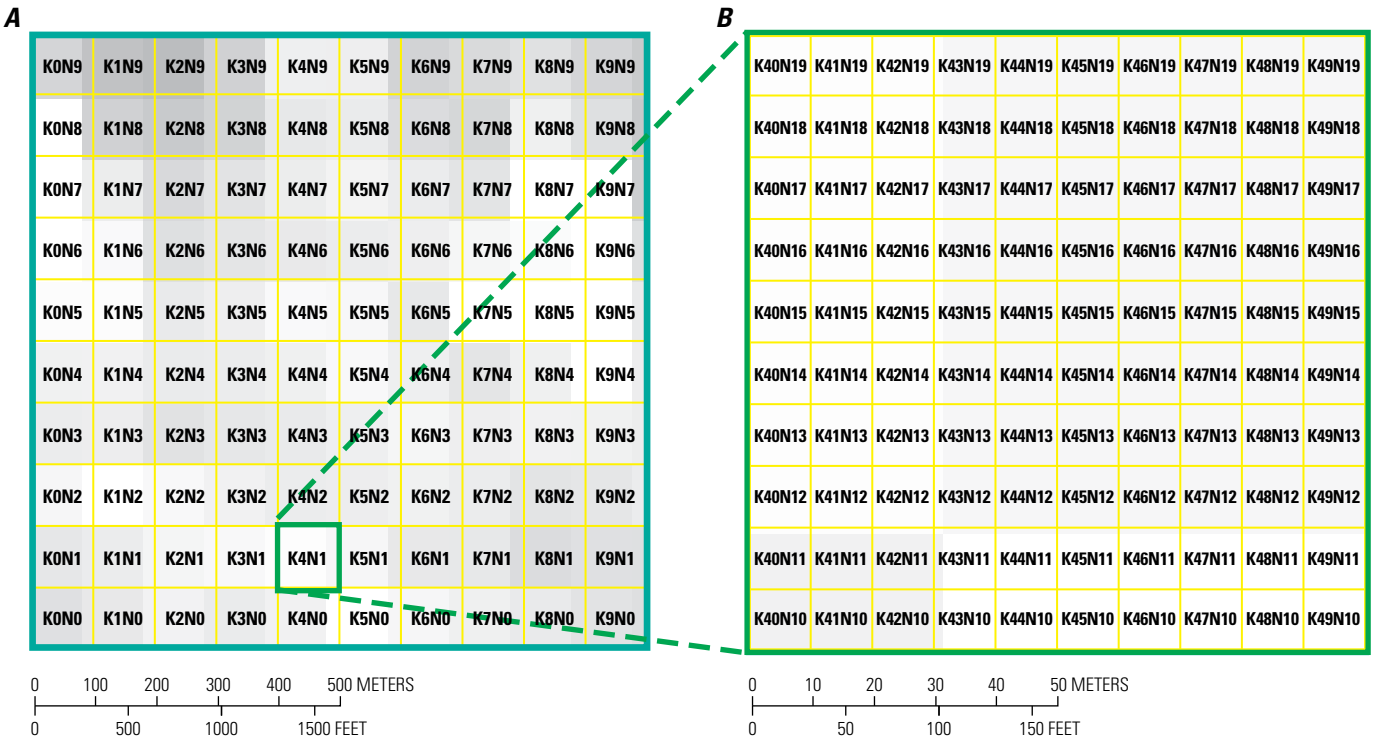


Figure 23. Map of the lunar south pole showing Lunar Grid Reference System (LGRS) 1-kilometer (km)-grid area BGQKN (teal box) with labeled 100-meter (m) grid (*A*) and LGRS 100-m-grid area BGQK4N1 (green box) with labeled 10-m grid (*B*). Note all LGRS coordinates are in Artemis Condensed Coordinates (ACC) format. Base from Lunar Reconnaissance Orbiter Camera (LROC) Team (2013).

As ACC truncates coordinates to a minimum precision of 10 m, there may be some precision loss during the inverse conversion. We have included functionality to process coordinates in ACC format to 1-m precision or use the polar zone and 25-km-grid area designators in the sample code provided in appendixes 1 and 2; thus, allowing anyone using this mapping system to accurately inversely convert their ACC positions to LGRS, the LTM or LPS systems, or latitude and longitude.

To redetermine the LGRS easting and northing positions, use the following equations:

$$E1k_{\text{num}} = 1,000 \times index\left(E1k\right), \tag{121}$$

and

$$N1k_{\text{num}} = 1,000 \times index\left(N1k\right). \tag{122}$$

Next, the easting and northing are summed with the remaining hundreds, tens, and, if available, one coordinate value of ACC as follows

$$E = E1k_{\text{num}} + E, \tag{123}$$

and

$$N = N1k_{\text{num}} + N, \tag{124}$$

where E and N are the remaining numeric portions of ACC. Note E and N will need to be zero-padded to replace any missing values that were lost during coordinate truncation. The remaining LGRS coordinate cannot be determined from coordinates in ACC format alone as the beginning portion of LGRS is not utilized; however, as the 25-km-grid area is assumed to be known and ACC is a relative coordinate, the LGRS inverse conversions can be applied to reconstruct the lunar global position. We recommend retaining this information in computer memory to fully determine the proper LGRS position. We also note, in the design and implementation of LGRS in ACC format, this information would be common knowledge.

LGRS Sample Conversion and Grid Generation Program

Preliminary software to complete the forward and inverse conversions of LGRS in ACC format is provided in appendix 1. Additional equations to create shapefiles of the ACC grids are provided in appendix 2.

Summary

This document describes the technical aspects behind a global lunar navigation standard for NASA's Artemis missions. Lunar surface navigation is based on the rectangular, topocentric coordinate systems that consist of the Lunar Transverse Mercator and Lunar Polar Stereographic systems, which support a Lunar Grid Reference System and a unique grid system, referred to as "Artemis Condensed Coordinates," for mission navigation purposes. These projected coordinate reference systems and grid system are similar in design to the terrestrial Universal Transverse Mercator and Universal Polar Stereographic systems and the Military Grid Reference System but differ and are designed to NASA's specifications for use on the Moon. Additionally, the Lunar Grid Reference System in Artemis Condensed Coordinate Format is designed similarly to the historic Army Map Service topographic orthophoto charts for Apollo mission lunar navigation (see https://www.lpi.usra.edu/resources/mapcatalog/topophoto/).

References Cited

Archinal, B.A., A'Hearn, M.F., II, Bowell, E., Conrad, A., Consolmagno, G.J., Courtun, R., Fukushima, T., Hestroffer, D., Hilton, J.L., Krasinsky, G.A., Neumann, G., Oberst, J., Seidemann, P.K., Stooke, P., Thomas, P.C., and Williams, I.P., 2011, Report of the IAU Working Group on Cartographic Coordinates and Rotational Elements—2009: Celestial Mechanics and Dynamical Astronomy, v. 109, p. 101–135, https://doi.org/10.1007/s10569-010-9320-4.

Archinal, B.A., Acton, C.H., A'Hearn, M.F., Conrad, A., Consolmagno, G.J., Duxbury, T., Hestroffer, D., Hilton, J.L., Kirk, R., Klioner, S.A., McCarthy, D., Meech, K., Oberst, J., Ping, J., Seidelmann, P.K., Tholen, D.J., Thomas, P.C., and Williams, I.P., 2018, Report of the IAU Working Group on Cartographic Coordinates and Rotational Elements—2015: Celestial Mechanics and Dynamical Astronomy, v. 130, article no. 22, 46 p., https://doi.org/10.1007/s10569-017-9805-5. [Also available at https://astropedia.astrogeology.usgs.gov/download/Docs/WGCCRE/WGCCRE2015reprint.pdf.]

Canters, F., 2002, Small-scale map projection design: London, Taylor & Francis, 352 p.

Davies, M.E., Abalakin, V.K., Cross, C.A., Duncombe, R.L., Masursky, H., Morando, B., Owen, T.C., Seidelmann, P.K., Sinclair, A.T., Wilkins, G.A., and Tjuflin, Y.S., 1980, Report of the IAU Working Group on Cartographic Coordinates and Rotational Elements of the Planets and Satellites: Celestial Mechanics, v. 22, p. 205–230, https://doi.org/10.1007/BF01229508.

Dennis, M.L., 2016, Ground truth—Low distortion map projections for engineering, surveying, and GIS *in* Proceedings of the Pipelines 2016 Conference, Kansas City, Mo., July 17–20, 2016: Reston, Va., Utility Engineering and Surveying Institute of the American Society of Civil Engineers.

Dennis, M.L., 2018, The State Plane Coordinate System—History, policy, and future directions: National Oceanic and Atmospheric Administration National Geodetic Survey Special Publication NOS NGS 13, 61 p., accessed May 1, 2023, at https://geodesy.noaa.gov/library/pdfs/SP_NOS_NGS_13.pdf.

Dennis, M.L., 2022, Map projections and local coordinate systems, chap. 3 *of* Gillins, D.T., Dennis, M.L., and Ng, A.Y, eds., Surveying and geomatics engineering—Principles, technologies, and applications: Reston, Va., American Society of Civil Engineers, Utility Engineering & Surveying Institute, p. 29–83, https://doi.org/10.1061/9780784416037.

Dennis, M.L., 2023, The future is here—Introducing the State Plane Coordinate System of 2022, *in* Proceedings of FIG Working Week 2023, Orlando, Florida, May 28–June 1, 2023: Copenhagen, International Federation of Surveyors, 15 p., accessed March 2024 at https://www.fig.net/resources/proceedings/fig_proceedings/fig2023/papers/cinema03/CINEMA03_dennis_12044.pdf.

Evans, N.F., 2014, British artillery fire control—Maps and survey: Royal Australian Artillery Historical Company web page, accessed February 2024 at https://www.artilleryhistory.org/gunners_past_and_present/obituaries/nigel_web/fc_maps.htm.

Folkner, W.M., Williams, J.G., and Boggs, D.H., 2008, The planetary and lunar ephemeris DE 421: California Institute of Technology, Jet Propulsion Laboratory Interoffice Memorandum 343R-08-003.

Hare, T.M., and Malapert, J.C., 2021, Standards proposal for 2021 to support planetary coordinate reference systems for open geospatial web services [abs.], *in* 5th Planetary Data and PSIDA, virtual, June 28–July 2, 2021, Program and Abstracts: Lunar and Planetary Institute, abstract no. 7012, accessed May 1, 2023, at https://www.hou.usra.edu/meetings/planetdata2021/pdf/7012.pdf.

Hirt, C., and Featherston, W., 2012, A 1.5 km-resolution gravity field model of the Moon: Earth, and Planetary Science Letters, v. 329–330, p. 22–30, https://doi.org/10.1016/j.epsl.2012.02.012.

Jet Propulsion Laboratory [JPL], 2005, Lunar constants and models document: Jet Propulsion Laboratory document D-32296, 66 p., accessed January 2024 at https://history.nasa.gov/alsj/lunar_cmd_2005_jpl_d32296.pdf. [Available at https://ssd.jpl.nasa.gov/doc/lunar_cmd_2005_jpl_d32296.pdf]

Jet Propulsion Laboratory [JPL], 2009, Planetary Data System standards reference, version 3.8: Jet Propulsion Laboratory document D-7669, part 2, accessed January 2024, at https://pds.nasa.gov/datastandards/pds3/standards/sr/StdRef_20090227_v3.8.pdf.

Jet Propulsion Laboratory [JPL], 2023, An overview of reference frames and coordinate systems in the SPICE context: Jet Propulsion Laboratory, Navigation and Ancillary Information Facility presentation, accessed January 2024 at https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Tutorials/pdf/individual_docs/17_frames_and_coordinate_systems.pdf.

Jones, E.M., ed., 2017, Apollo 11–17 lunar surface journals: National Aeronautics and Space Administration web page, accessed March 2024 at https://www.nasa.gov/history/alsj/main.htm.

Karney, C.F.F., 2011, Transverse Mercator with an accuracy of a few nanometers: Journal of Geodesy, v. 85, no. 8, p. 475–485, https://doi.org/10.1007/s00190-011-0445-3.

Kinsler, D.K., 1975, User guide to 1:250,000 scale lunar maps: Lunar Science Institute contribution no. 206, accessed May 1, 2023, at https://core.ac.uk/download/pdf/222788082.pdf.

Krüger, L., 1912, Konforme abbildung des Erdellipsoids in der ebene [Conformal mapping of the ellipsoidal Earth to the plane]: Royal Prussian Geodetic Institute, New Series, v. 52, 172 p., https://doi.org/10.2312/GFZ.b103-krueger28.

Lee, L.P., 1945, The transverse Mercator projection of the spheroid: Empire Survey Review, v. 8 no. 61, p. 277–278.

Lunar Reconnaissance Orbiter mission and Lunar Geodesy and Cartography Working Group [LRO mission and LGCWG], 2008, A standardized lunar coordinate system for the Lunar Reconnaissance Orbiter and lunar datasets (ver. 5): LRO mission and LGCWG White Paper, accessed March 2023 at https://lunar.gsfc.nasa.gov/library/LunCoordWhitePaper-10-08.pdf.

Lunar Reconnaissance Orbiter Camera [LROC] Team, 2013, Moon LRO LROC WAC global morphology mosaic 100 m (ver. 3): U.S. Geological Survey, Astropedia—Lunar and Planetary Cartographic Catalog, accessed March 2023 at https://astrogeology.usgs.gov/search/map/Moon/LRO/LROC_WAC/Lunar_LRO_LROC-WAC_Mosaic_global_100m_June2013. [Available at https://astrogeology.usgs.gov/search/map/moon_lro_lroc_wac_global_morphology_mosaic_100m.]

Lunar Orbiter Laser Altimeter [LOLA] Science Team, 2021, Moon LRO LOLA DEM 118 m, [digital elevation model]: U.S. Geological Survey, Astropedia—Lunar and Planetary Cartographic Catalog, accessed March 2023 at https://astrogeology.usgs.gov/search/details/Moon/LRO/LOLA/Lunar_LRO_LOLA_Global_LDEM_118 m_Mar2014/cub. [Available at https://astrogeology.usgs.gov/search/map/moon_lro_lola_dem_118m.]

Ma, C., Arias, E.F., Eubanks, T.M., Fey, A.L., Gontier, A.M., Jacobs, C.S., Sovers, O.J., Archinal, B.A., and Charlot, P., 1998, The International Celestial Reference Frame as realized by very long baseline interferometry: Astronomical Journal, v. 116, p. 516–546.

Markič, S., Donaubauer, A., and Borrmann, A., 2018, Enabling geodetic coordinate reference systems in building information modeling for infrastructure, *in* 17th International Conference on Computing in Civil and Building Engineering, Tampere, Finland, June 5–7, 2018, Proceedings: Montreal, International Society for Computing in Civil and Building Engineering, 8 p., https://publications.cms.bgu.tum.de/2018_Markic_ICCCBE.pdf.

McClernan, M.T., Archinal, B.A., Skinner, J.A., and Buban, H.C., 2023, Proposed geographic coordinate system for lunar surface navigation and the Artemis missions [abs.], *in* 6th Planetary Data Workshop, Flagstaff, Ariz., June 26–28, 2023, Technical Program: Lunar and Planetary Science Institute, abstract no. 7023, 2 p., https://www.hou.usra.edu/meetings/planetdata2023/pdf/7023.pdf.

National Aeronautics and Space Administration [NASA], 1969, LM lunar surface maps: National Aeronautics and Space Administration, part SKB32100081-371, s/n 1002, accessed March 2023 at https://nasa.gov/wp-content/uploads/static/history/alsj/a12/a12_lsm.pdf.

National Aeronautics and Space Administration [NASA], 1973, Lunar cartographic dossier [Schimerman, L.A., ed.]: St. Louis AFS, Mo., Defense Mapping Agency, v. 1., 359 p.

National Aeronautics and Space Administration [NASA], 2008, A standardized Lunar Coordinate System for the Lunar Reconnaissance Orbiter and lunar datasets, version 5: Greenbelt, Md., National Aeronautics and Space Administration, Goddard Space Flight Center, LRO Project and LGCWG White Paper, 13 p.

National Geospatial-Intelligence Agency [NGA], 2014a, Department of Defense World Geodetic System 1984—Its definition and relationships with local geodetic systems, version 1.0.0: National Geospatial-Intelligence Agency Standardization Document NGA. STND.0036_1.0.0_WGS84, 208 p.

National Geospatial-Intelligence Agency [NGA], 2014b, The universal grids and the transverse Mercator and polar stereographic map projections, version 2.0.0: National Geospatial-Intelligence Agency Standardization Document NGA.SIG.0012_2.0.0_ UTMUPS, 86 p.

Palombo, S., 2021, Military Grid Reference System (MGRS): Esri Story Map, accessed February 2024 at https://storymaps.arcgis.com/stories/842edf2b4381438b9a4edefed124775b.

Park, R.S., Folkner, W.M., Williams, J.G., Boggs, D.H., 2021, The JPL planetary and lunar ephemerides DE440 and DE441: The Astronomical Journal, v. 161, no. 3, 15 p., https://doi.org/10.3847/1538-3881/abd414.

Redfearn, J.C.B., 1948, Transverse Mercator formulae: Empire Survey Review, v. 9, no. 69, p. 318–322, https://doi.org/10.1179/sre.1948.9.69.318.

Snay, R.A., 2012, Evolution of NAD 83 in the United States—Journey from 2D toward 4D: Journal of Surveying Engineering, v. 138, no. 4, p. 161–171.

Snyder, J.P., 1987, Map projections—A working manual: U.S. Geologic Survey Professional Paper 1395, 397 p.

Song, Y., Lee, D., Kim, Y., Bae, J., Park, J., Hong, S., Kim, D., and Lee, S., 2021, Practical algorithms on lunar reference frame transformations for Korea Pathfinder Lunar Orbiter flight operation: Journal of Astronomy and Space Science Technical Paper, v. 38, no. 3, p. 185–192, https://doi.org/10.5140/JASS.2021.38.3.185.

Stem, J. E., 1990, State Plane Coordinate System of 1983: National Oceanographic and Atmospheric Administration, Manual NOS NGS 5, 130 p., accessed May 1, 2023, at https://geodesy.noaa.gov /library/pdfs/NOAA_Manual_NOS_NGS_0005.pdf.

Thomas, P. D. 1952, Conformal projections in geodesy and cartography: U.S. Geodetic Survey Special Publication no. 251., 142 p.

Williams, J.G., Konopliv, A.S., Boggs, D.H., Park, R.S., Yuan, D., Lemoine, F.G., Goossens, S., Mazarico, E., Nimmo, F., Weber, R.C., Asmar, S.W.; Melosh, H.J., Neumann, G.A., Phillips, R.J., Smith, D.E. Solomon, S.C., Watkins, M.M., Wieczorek, M.A., Andrews-Hanna, J.C., Head, J.W., Kiefer, W.S., Matsuyama, I., McGovern, P.J., Taylor, G.J., and Zuber, M.T., 2014, Lunar interior properties from the GRAIL mission: Journal of Geophysical Research—Planets, v. 119, p. 1546–1578, https://doi.org/10.1002/2013JE004559.

Appendixes 1–3

Appendix 1. Preliminary Lunar Grid Reference System Coordinate Conversion Program

Foreword

This appendix contains a preliminary version of a lunar coordinate conversion program developed to facilitate seamless transformation between the associated lunar projected coordinate reference systems (PCRS) and the Lunar Grid Reference System (LGRS). This software aims to provide readers with a preliminary release of the LGRS to help facilitate testing, review, and future improvements. Input coordinates can include various formats such as planetocentric latitude and longitude, Lunar Transverse Mercator (LTM), Lunar Polar Stereographic (LPS) and the LGRS. The program currently supports basic conversions between these systems, ensuring accuracy and reliability in transforming geographic coordinates to projected coordinate systems and vice versa. Future iterations of the software will include additional functionalities such as datum transformations between the lunar principal axis system and a software refactor. Code operations and the Python code of the Coordinate Conversion program are provided below.

Code Operation

The use of the coordinate system conversion coded is designed to be simple. The generic way to call the conversion program is as follows:
./LGRS_Coordinate_Conversion_mk7.py {Input format}2{Output Format} {Coordinate Values}.
The input formats are as follows:

- LatLon—Latitude–longitude pair between latitudes −82° and 82°
- PolarLatLon—Polar latitude–longitude pair between latitudes −90° and −80° or latitudes 80° and 90°
- LTM—Lunar Transverse Mercator coordinates
- LPS—Lunar Polar Stereographic coordinates
- LGRS—Lunar Grid Reference System coordinates
- LGRS_ACC—Lunar Grid Reference System coordinates in Artemis Condensed Coordinates (ACC) format
- PolarLGRS—Lunar Grid Reference System coordinates in polar regions
- PolarLGRS_ACC—Lunar Grid Reference System coordinates in ACC format in polar regions

The output formats are the same as the input formats and include two additional grids:

- LatLon
- PolarLatLon
- LTM
- LPS
- LGRS
- LGRS_ACC
- PolarLGRS
- PolarLGRS_ACC
- ACC—Lunar Grid Reference System coordinates in ACC format truncated to 6 characters
- PolarACC—Lunar Grid Reference System coordinates in ACC format in polar regions truncated to 6 characters

Note that ACC and PolarACC cannot be inverted as they are truncated to 6 characters to meet National Aeronautics and Space Administration (NASA) design specifications.

The input coordinates may be provided to the program in two formats, compressed format and uncompressed. Then compressed format has no functional delimiters or spacing and must be truncated to at least 1-m precision. The uncompressed

format requires a space be utilized between each coordinate portion on the command line prior to its execution. No truncation is required in this format and submeter decimal precision can be processed allowing for limited coordinate truncation error. For example, an LTM coordinate is composed of four components, zone number, *Z*; hemisphere, *H*; easting, *E*; and northing, *N*; in compressed format the coordinate is formatted as ZZH00000E00000N. In uncompressed format the coordinate is formatted as ZZ H 000000.0 00000.0. A processing flag is set at the beginning of the program that will specify this formatting for the input and assumed output format. Additionally, the grid system truncation level is set at the beginning of the program. See LGRS conversion equations for more details.

Please contact the U.S. Geological Survey or the Artemis Geospatial Data Team for support and assistance.

Preliminary LGRS Coordinate Conversion Python Program

```
# !/bin/env python3
# ======================================================================
‘’’ PROGRAM INFORMATION
Program:  LGRS_Coordinate_Conversion_mk7.py
Language: Python 3.11.4
Author:   Mark T. McClernan
Created:  March, 2023   (Python 3.6.5)
Modified: February, 2024 (Python 3.11.4)
IDE:      Visual Studio Code (v1.18.1 x86)

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
PROGRAM DESCRIPTION:
   This program performs forward and inverse conversions from
   lunar planetocentric latitude and longitude to each of the
   following lunar coordinate systems:
      1. Lunar Transverse Mercator (LTM) system, 8° zones;
      2. Lunar Polar Stereographic (LPS);
      3. Lunar Grid Reference System (LGRS);
      4. LGRS in Artemis Condensed Coordinate Format (ACC)
      and
      5. LGRS ACC coordinates limited to 6 characters.

   Algorithms are modified for lunar use and provided from Karney
   (2010), Snyder (1987), and NGA (2014). See References.

   Main modifications to projection and equations from references:
      1. Change flattening value to zero to support a lunar spheroid.
      2. Utilizing 8° transverse Mercator zone.
      3. Map projection parameters are all updated to support Lunar use
      4. Grid systems use 25km and 1km.

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
HOW TO USE THIS CODE:

   This software is designed to run via the command line, where an
   input coordinate is provided to the software and the converted
   format is returned in the specified format.

   The program is called from the command line with the sample
   command

   ./LGRS_Coordinate_Conversion_mk7.py {form_in}2{form_out} {Coordinate}

   Warnings and argument checks are included to make sure the converted
```

format is correct and will terminate the program if the conversion
is not able to take place.

form_in can be LatLon, LTM, LPS, LGRS, PolarLGRS, LGRS_ACC, and PolarLGRS_ACC
form_out can be LatLon, LTM, LPS, LGRS, PolarLGRS, LGRS_ACC, PolarLGRS_ACC,
ACC or Polar_ACC

The coordinate variable has two input modes, condensed and noncondensed.
If not condensed the coordinate needs to be spaced out for each
coordinate component, the program is looking for a specific number
of coordinates and if the correct number is not met the program will
terminate.

LPS: H, E, N
LTM: Z, H, E, N
PolarLGRS: latBand, 25kmE, 25kmN, E, N
LGRS: lonBand, latBand, 25kmE, 25kmN, E, N
PolarLGRS_ACC: latBand, 25kmE, 25kmN, 1kmE, 1kmN E, N
LGRS_ACC: lonBand, latBand, 25kmE, 25kmN, 1kmE, 1kmN E, N

Easting and northing values do not need to be truncated and can be
submitted as floating point values. A space is required between each
coordinate value, no special delimiters are to be used.

If condensed the input coordinate must be truncated to at least 1m and
written with no spaces. A delimiter of (E) for easting and (N) for
northing is required to input an LTM or LPS coordinate.
LTM: 23S250000E1894139N
LPS: S500000E741523N

LGRS and LGRS coordinates do not require an easting or northing
delimiter in condensed mode. Additionally, this import format is
not available for latitude and longitude inputs.

The condensed condition is set via a boolean value at the beginning of
the program
condensed = True or condensed = False

The condensed variable also changes the output, if condensed is specified,
a condensed argument will be returned. If not condensed the output
coordinate will be spaced out depending on the number of variables of the
final coordinate.

Two other variables can be set to change the program's processing
behavior. trunc_val specifies the degree at which the coordinate
is truncated to. 1 for one place, 10 for tens, etc... this value must be
a multiple of 10 or the program will terminate. If trunc_val is 0
no truncation will occur and a floating point is returned.

The info returns processing errors and the processing time of the
conversion has taken

Example program calls: # settings following after hash,
Note round-off error from truncation

LTM 1:

```
  in
    ./LGRS_Coordinate_Conversion_mk7.py LatLon2LTM 20.0 0.0
    # trunc_val=1, condensed=True
  out
    23N250000E0605860N

LTM 2:
  in
    ./LGRS_Coordinate_Conversion_mk7.py LatLon2LTM 20.0 0.0
    # trunc_val=0, condensed=False
  out
    23 N 250000.0 605860.5414745066

LPS 1:
  in
    ./LGRS_Coordinate_Conversion_mk7.py LatLon2LPS -80.0 -135.0
    # trunc_val=1, condensed=True
  out
  S286325E286325N

LPS 2:
  in
    ./LGRS_Coordinate_Conversion_mk7.py LatLon2LPS -80.0 -135.0
     # trunc_val=0, condensed=False
  out
    S 286325.3596121004 286325.3596121003


LGRS 1:
  in
    ./LGRS_Coordinate_Conversion_mk7.py LTM2LGRS 23N250000E0605860N
    # trunc_val=1, condensed=True
  out
    23QFK0000005860

LTM 3:
  in
    ./LGRS_Coordinate_Conversion_mk7.py LGRS2LTM 23QFK0000005860
    # trunc_val=1, condensed=True

  out
    23N250000.0E605860.0N

Polar LGRS 1:
  in
    ./LGRS_Coordinate_Conversion_mk7.py LatLon2PolarLGRS -80.0 -135.0
    # trunc_val=1, condensed=True

  out
    ATF0421604216

  LatLon 4:
  in
    ./LGRS_Coordinate_Conversion_mk7.py PolarLGRS2LatLon ATF0421604216
    # trunc_val=1, condensed=True
  out
```

```
        -81.99995863117528°-135.0° (Round off error)

    LGRS_ACC 1:
      in
        ./LGRS_Coordinate_Conversion_mk7.py LTM2LGRS_ACC 23N250000E0605860N
      out
        23QFK-000E860

    LTM 4:
      in
        ./LGRS_Coordinate_Conversion_mk7.py LGRS_ACC2LTM 23QFK-000E860
      out
        23N250000.0E605860.0N

    PolarLGRS_ACC 1:
      in
        ./LGRS_Coordinate_Conversion_mk7.py LatLon2PolarLGRS_ACC -82.0 -135.0
        # trunc_val=1, condensed=True
      out
        ATFD216D216

    LatLon 5:
      in
        ./LGRS_Coordinate_Conversion_mk7.py PolarLGRS_ACC2LatLon ATFD216D216
      out
        -81.99995863117528°-135.0° (Round off error)

    To produce the ACC grid for lunar surface navigation set trunc_val to
    10 or specify ACC or Polar_ACC. There is no inverse conversion, as the
    coordinate is relative and truncated to report in a 25km grid zone
    to a precision of 10m.

    Polar_ACC 1
      in
        ./LGRS_Coordinate_Conversion_mk7.py LatLon2Polar_ACC -82.0 -135.0
      out
        D21D21

    Polar_ACC
      in
        ./LGRS_Coordinate_Conversion_mk7.py LTM2ACC 23N250000E0605860N
      out
        -00E86

    See the testing script for more examples.

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
PROGRAM SECTIONS (can search for each using "SECTION #"):
    SECTION 0: INITIALIZE PROGRAM AND CALL EXTERNAL LIBRARIES
    SECTION 1: REFERENCE SURFACE PARAMETERS
        SECTION 1.1:   parameters needed for conversion equations
            SECTION 1.1.1: parameters needed for Karney-Kruger (2010)
                    based Lunar Transverse Mercator (LTM) system.
            SECTION 1.1.2: parameters needed for Snynder (1987) and the Lunar
                    Polar Stereographic System
            SECTION 1.1.3: Parameters Needed for Lunar Grid Reference
```

```
            System (LGRS)
        SECTION 1.1.4: parameters needed for LGRS in Artemis
            Condensed Coordinate (ACC) Format
        SECTION 1.1.5: Patterns needed to read in condensed coordinates

    SECTION 2: FUNCTION LIBRARY
        SECTION 2.1: Convert to LTM from planetocentric Lat and Lon
        SECTION 2.2: Convert to planetocentric Lat and Lon from LTM
        SECTION 2.3: Calculate meridional radius of curvature
        SECTION 2.4: Calculate distance from pole to latitude parallel
        SECTION 2.5: Calculate polar stereographic ellipsoidal X coordinate
        SECTION 2.6: Calculate polar stereographic ellipsoidal Y coordinate
        SECTION 2.7: Calculate polar stereographic spherical scale error
        SECTION 2.8: Calculate polar stereographic spherical X coordinate
        SECTION 2.9: Calculate polar stereographic spherical Y coordinate
        SECTION 2.10: Convert to LPS from planetocentric Lat and Lon
        SECTION 2.11: Inverse calculate meridional radius of curvature
        SECTION 2.12: Calculate conformal latitude
        SECTION 2.13: Recover Latitude on an ellipsoid without iteration
        SECTION 2.14: Recover Longitude on an ellipsoid
        SECTION 2.15: Calculate distance from projection pole to parallel
        SECTION 2.16: Calculate great circle distance from pole to parallel
        SECTION 2.17: Recover latitude on a sphere
        SECTION 2.18: Recover longitude on a sphere
        SECTION 2.19: Convert to planetocentric Lat and Lon from LPS
        SECTION 2.20: Convert to LGRS from LTM
        SECTION 2.21: Convert to LGRS from LPS
        SECTION 2.22: Convert to LTM from LGRS
        SECTION 2.23: Convert to LPS from LGRS
        SECTION 2.24: Convert to LGRS in ACC format from LTM
        SECTION 2.25: Convert to LGRS in ACC format from LPS
        SECTION 2.26: Convert to LTM form LGRS in ACC format
        SECTION 2.27: Convert to LPS from LGRS in ACC format
        SECTION 2.28: Convert planetocentric latitude to colatitude
        SECTION 2.29: Convert planetocentric colatitude to latitude
        SECTION 2.30: Convert planetocentric longitude to colongitude
        SECTION 2.31: Convert planetocentric colongitude to longitude
        SECTION 2.32: Convert degrees to decimal minutes
        SECTION 2.33: Convert decimal minutes to degrees
        SECTION 2.34: Convert degrees to decimal seconds
        SECTION 2.35: Convert decimal seconds to degrees
        SECTION 2.36: Formula to truncate coordinates

        SECTION 3: MAIN PROGRAM
            SECTION 3.1: Importing system variables for processing
            SECTION 3.2: Converting Coordinates
            SECTION 3.2.1: LATLON CONVERSIONS
                SECTION 3.2.2: LTM CONVERSIONS
                SECTION 3.2.3: LPS CONVERSIONS
                SECTION 3.2.4: LGRS CONVERSIONS
                SECTION 3.2.4: LGRS ACC CONVERSIONS
            SECTION 3.3: Exporting data
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# References
```

Karney, C.F.F. (2010), Transverse Mercator with an accuracy of a few

nanometers Journal of Geodesy, 85(8), 475–485,
https://doi.org/10.48550/arXiv.1002.1417.

NGA (2014), The Universal Grids and the Transverse Mercator and Polar
Stereographic Map Projections, in National Geospatial-Intelligence
Agency (NGA) Standardization Document, NGA.SIG.0012_2.0.0_UTMUPS.

Snyder, J. P. (1987). Map Projections: A Working Manual, U.S.
Geological Survey Professional Paper 1395, Washington, DC: United
States Government Printing Office,
https://pubs.usgs.gov/pp/1395/report.pdf.

```
‘’’
# ============================================================================
# SECTION 0: INITIALIZE PROGRAM AND CALL EXTERNAL LIBRARIES


import os               # used as system utilities
import sys
                            # os -> Python 3.11.4
                            # sys 3.11.4

import numpy as np        # To perform mathematical calculations
                            # np 1.25.2

import datetime          # To perform time monitoring
                            # datetime -> Python 3.11.4

import re               # for input coordinate parsing
                    # re -> ‘2.2.1’

ProgName = “LGRS_Coordinate_Conversion_mk7”

# ============================================================================
# SECTION 1: REFERENCE SURFACE PARAMETERS
# Geodetic function constants for the Moon, note all projection origins
#     are determined in their associated function



def initialize_LGRS_function_globals():

    global FalseEasting, FalseNorthing, FalseEasting_polar
    global FalseNorthing_polar, k0, k0_polar, ZoneWidth, a, f, b, e

    # Map projection False Origins
    # False Origins for LTM

    FalseEasting = 250E3
    FalseNorthing = 2500E3

    # False Origins for LPS
    FalseEasting_polar = 500E3
    FalseNorthing_polar = 500E3

    # map projection scale factors
    k0 = .999        # transverse Mercator (LTM)
```

```
    k0_polar = .994   # polar stereographic (LPS)

    # Transverse Mercator zone width from the central meridian
    ZoneWidth = 4.

    # Lunar spheroid parameters
    a = 1737400       # Semi-major radius
    f = 0             # Geometric flattening
    b = a * (1 - f)       # Semi-minor radius
    e = (f * (2 - f))**.5  # First eccentricity

    # Ensure a full double is processed and printed
    np.set_printoptions(precision=20)

# ----------------------------------------------------------------------
# SECTION 1.1: parameters needed for conversion equations

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# SECTION 1.1.1: Parameters needed for Karney-Kruger (2010) based Lunar
#               Transverse Mercator (LTM) system.

    # 3rd ellipsoidal flattening
    global n, n2, n3, n4, n5, n6

    n = f / (2 - f)       # can use n = (a - b)/(a + b).

    # construct powers of n to order 6
    n2 = n*n
    n3 = n*n2
    n4 = n*n3
    n5 = n*n4
    n6 = n*n5

    # rectifying radius of conformal spheres to order 6
    global A
    A = a/(1 + n) * (1 + (1/4)*n2 + (1/64)*n4 + (1/256)*n6)

    # construct alpha coefficients (order 6) for forward conversion
    global ap2, ap4, ap6, ap8, ap10, ap12

    ap2 = (1/2)*n - (2/3)*n2 + (5/16)*n3 + (41/180)*n4 - (127/288)*n5 \
              + (7891/37800)*n6
    ap4 = (13/48)*n2 - (3/5)*n3 + (557/1440)*n4 + (281/630)*n5 \
                - (1983433/1935360)*n6
    ap6 = (61/240)*n3 - (103/140)*n4 + (15061/26880)*n5 + (167603/181440)*n6
    ap8 = (49561/161280)*n4 - (179/168)*n5 + (6601661/7257600)*n6
    ap10 = (34729/80640)*n5 - (3418889/1995840)*n6
    ap12 = (212378941/319334400)*n6

    # construct beta coefficients (order 6) for inverse conversion
    global bt2, bt4, bt6, bt8, bt10, bt12

    bt2 = (1/2)*n - (2/3)*n2 + (37/96)*n3 - (1/360)*n4 - (81/512)*n5 \
              + (96199/604800)*n6
    bt4 = (1/48)*n2 + (1/15)*n3 - (437/1440)*n4 + (46/105)*n5 \
                - (1118711/3870720)*n6
```

```
    bt6 = (17/480)*n3 - (37/840)*n4 - (209/4480)*n5 + (5569/90720)*n6
    bt8 = (4397/161280)*n4 - (11/504)*n5 - (830251/7257600)*n6
    bt10 = (4583/161280)*n5 - (108847/3991680)*n6
    bt12 = (20648693/638668800)*n6


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# SECTION 1.1.2: parameters needed for Snyder (1987) Lunar Polar
#                Stereographic

# coefficients needed for iteration during the inverse transform
# see Snyder (1987)

    global e2, e4, e6, e8

    e2 = e*e
    e4 = e2*e2
    e6 = e4*e2
    e8 = e6*e2

    global C2, C4, C6, C8

    C2 = e2/2 + 5*e4/24 + e6/12 + 13*e8/360
    C4 = 7*e4/48 + 29*e6/240 + 811*e8/11520
    C6 = 7*e6/120 + 81*e8/1120
    C8 = 4279*e8/161280

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# SECTION 1.1.3: parameters needed for Lunar Grid Reference System (LGRS)

    global latBands, e25kLetters, n25kLetters, e25kLetters_polar, \
        n25kLetters_polar
    # Latitude bands C..X 8° each, covering 88°S to 88°N
    # letters duplicated for oversized zones
    latBands = 'CCDEFGHJKLMNPQRSTUVWXX'

    # Letters used for the 25km LGRS in the LTM portion
    e25kLetters = ['ABCDEFGHJK']
    n25kLetters = ['ABCDEFGHJKLMNPQRSTUV', 'FGHJKLMNPQRSTUVABCDE',
                   'LMNPQRSTUVABCDEFGHJK']

    # Letters used for the 25km LGRS in the LPS portion
    e25kLetters_polar = 'ABCDEFGHJKLMNPQRSTUVWXYZ'
    n25kLetters_polar = 'ABCDEFGHJKLMNPQRSTUVWXYZ'

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# SECTION 1.1.4: Parameters needed for LGRS in Artemis Condensed
#                Coordinate (ACC) Format
    global e1kmLetters, n1kmLetters

    # Letters used for the 1km grid used in LGRS in condensed format
    e1kmLetters = '-ABCDEFGHJKLMNPQRSTUVWXYZ'  # "-" is a zero character
    n1kmLetters = '-ABCDEFGHJKLMNPQRSTUVWXYZ'

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# SECTION 1.1.5: Patterns needed to read in condensed coordinates
```

```
    global LTM_pattern, LPS_pattern, LGRS_pattern
    global PolarLGRS_pattern, LGRS_ACC_pattern, PolarLGRS_ACC_pattern

    # supports floating or truncated integer values
    LTM_pattern = r'(\d+)([NS]{1})(\d+(?:\.\d+)?|\.\d+)(E)(\d+(?:\.\d+)?|\.\d+)(N)'
    LPS_pattern = r'([NS]{1})(\d+(?:\.\d+)?|\.\d+)(E)(\d+(?:\.\d+)?|\.\d+)(N)'

    # supports only truncated values (this is ok; non-compressed
    # Coordinates can be read in as a full floating point)
    LGRS_pattern = r'(\d+)([A-Z]{1})([A-Z]{1})([A-Z]{1})(\d{5})(\d{5})'
    PolarLGRS_pattern = r'([A-Z]{1})([A-Z]{1})([A-Z]{1})(\d{5})(\d{5})'
    LGRS_ACC_pattern = r'(\d+)([A-Z]{1})([A-Z]{1})([A-Z]{1})(.{1})(\d+)(.{1})(\d+)'
    PolarLGRS_ACC_pattern = r'([A-Z]{1})([A-Z]{1})([A-Z]{1})(.{1})(\d+)(.{1})(\d+)'

# ========================================================================
# SECTION 2: FUNCTION LIBRARY
    # SECTION 2.1: Convert to LTM from planetocentric Lat and Lon
    # SECTION 2.2: Convert to planetocentric Lat and Lon from LTM
    # SECTION 2.3: Calculate meridional radius of curvature
    # SECTION 2.4: Calculate distance from pole to latitude parallel
    # SECTION 2.5: Calculate polar stereographic ellipsoidal X coordinate
    # SECTION 2.6: Calculate polar stereographic ellipsoidal Y coordinate
    # SECTION 2.7: Calculate polar stereographic spherical scale error
    # SECTION 2.8: Calculate polar stereographic spherical X coordinate
    # SECTION 2.9: Calculate polar stereographic spherical Y coordinate
    # SECTION 2.10: Convert to LPS from planetocentric Lat and Lon
    # SECTION 2.11: Inverse calculate meridional radius of curvature
    # SECTION 2.12: Calculate conformal latitude
    # SECTION 2.13: Recover Latitude on an ellipsoid without iteration
    # SECTION 2.14: Recover Longitude on an ellipsoid
    # SECTION 2.15: Calculate distance from projection pole to parallel
    # SECTION 2.16: Calculate great circle distance from pole to parallel
    # SECTION 2.17: Recover latitude on a sphere
    # SECTION 2.18: Recover longitude on a sphere
    # SECTION 2.19: Convert to planetocentric Lat and Lon from LPS
    # SECTION 2.20: Convert to LGRS from LTM
    # SECTION 2.21: Convert to LGRS from LPS
    # SECTION 2.22: Convert to LTM from LGRS
    # SECTION 2.23: Convert to LPS from LGRS
    # SECTION 2.24: Convert to LGRS in ACC format from LTM
    # SECTION 2.25: Convert to LGRS in ACC format from LPS
    # SECTION 2.26: Convert to LTM form LGRS in ACC format
    # SECTION 2.27: Convert to LPS from LGRS in ACC format
    # SECTION 2.28: Convert planetocentric latitude to colatitude
    # SECTION 2.29: Convert planetocentric colatitude to latitude
    # SECTION 2.30: Convert planetocentric longitude to colongitude
    # SECTION 2.31: Convert planetocentric colongitude to longitude
    # SECTION 2.32: Convert degrees to decimal minutes
    # SECTION 2.33: Convert decimal minutes to degrees
    # SECTION 2.34: Convert degrees to decimal seconds
    # SECTION 2.35: Convert decimal seconds to degrees
    # SECTION 2.36: Formula to truncate coordinates

# ------------------------------------------------------------------------
# SECTION 2.1: Convert to LTM from planetocentric Lat and Lon
```

```
def toLTM(lon, lat, zone=None, trunc_val=1, process_errors=True):
    """Converts Planetocentric LatLon degree values to LTM easting
       and northing coordinates. Latitude and Longitude are the only
       Inputs required. Optionally, the zone does not have to be
       specified. If specified the central meridian will be
       calculated for the specified zone. An option is available to
       truncate the coordinates. 1 m is the default needed for LTM
       coordinates specifically; however, there will be round off with
       inverse conversion or additional conversions to other coordinate
       systems. For grid systems set to 0 for no truncation.

    Parameters:
    lon             Longitude              (float): degrees
    lat             Latitude               (float): degrees
    zone            LTM Zone               (int,float,NoneType),scaler
    lam0            Central Meridian to Zone(float): degrees
    trunc_val       Coordinate Precision    (int,float): meters
    process_errors  Processing flag         (logical): boolean

    Returns:
    zone            LTM zone               (int) scaler
    h               hemisphere             (str) unitless
    E               Eastings               (float,str) meters
    N               Northings              (float,str) meters

    Raises:
    ValueError: If coordinates are not within a latitude 90 -90,
                longitude -180-180 range.
            User challenged if value is in the extended range"""
    if process_errors:
        if lat > 90:
            raise ValueError("Operation aborted: Latitude exceeds 90°.")
        elif lat < -90:
            raise ValueError("Operation aborted: Latitude less than -90°.")
        elif lon > 180:
            raise ValueError("Operation aborted: Longitude exceeds 180° "
                             " Try converting from 0°-360° to "
                             "-180°-180° longitude range.")
        elif lon < -180:
            raise ValueError("Operation aborted: Longitude is less than 180°")
        elif (lat >= -90 and lat < -82) or (lat <= 90 and lat > 82):
            choice = input("Latitude is in the extended between +-82° "
                           "and +-90° degrees. Do you wish to force the "
                           "conversion? LGRS is not supported in this region."
                           "(y/n): ")
            if choice.lower() != 'y':
                raise ValueError("Operation aborted: Latitude outside of "
                                 "LTM projection areas")
        elif (lat >= -82 and lat < -80) or (lat <= 82 and lat > 80):
            choice = input("Latitude is in the extended between +-80° "
                           "and +-82° degrees. Do you wish to continue the "
                           "conversion? (y/n): ")
            if choice.lower() != 'y':
                raise ValueError("Operation aborted: Latitude outside of "
                                 "LTM projections areas")
```

```
# determine correct hemisphere for coordinate and processing
if lat >= 0.:
   h = 'N'
else:
   h = 'S'

# determine LTM zone if none is given
if zone == None:    # if no zone is given

   # longitudinal zone
   zone = np.floor((np.mean(lon) + 180)/(ZoneWidth * 2)) + 1

   if zone > 45:  # 180° values sometimes come up as 46.
                  # We assign it back to zone 1
      zone -= 45

# longitude of central meridian
lam0 = np.deg2rad((zone - 1) * (ZoneWidth * 2) - 180. + (ZoneWidth))

# convert from degrees to radians
phi = np.deg2rad(lat)  # latitude from equator
lam = np.deg2rad(lon) - lam0  # longitude ± from central meridian

# precompute simple longitude values for speed
coslam = np.cos(lam)
sinlam = np.sin(lam)
tanlam = np.tan(lam)

# precompute latitude values
tanphi = np.tan(phi)  # prime (_p) indicates angles on the conformal sphere

# Compute conformal latitude
sigma = np.sinh(e*np.arctanh(e*tanphi / (1 + tanphi**2)**.5))
T_p = tanphi * (1 + sigma*sigma)**.5 - sigma * (1 + tanphi*tanphi)**.5

# compute Gauss Schreiber coordinates
E_p = np.arctan2(T_p, coslam)
N_p = np.arcsinh(sinlam / (T_p*T_p + coslam*coslam)**.5)

# Compute unscaled X,Y coordinates
Ecoordonate = (E_p
            + ap2 * np.sin(2*E_p) * np.cosh(2*N_p)
            + ap4 * np.sin(4*E_p) * np.cosh(4*N_p)
            + ap6 * np.sin(6*E_p) * np.cosh(6*N_p)
            + ap8 * np.sin(8*E_p) * np.cosh(8*N_p)
            + ap10 * np.sin(10*E_p) * np.cosh(10*N_p)
            + ap12 * np.sin(12*E_p) * np.cosh(12*N_p))

Ncoordonate = (N_p
            + ap2 * (np.cos(2*E_p)) * (np.sinh(2*N_p))
            + ap4 * (np.cos(4*E_p)) * (np.sinh(4*N_p))
            + ap6 * (np.cos(6*E_p)) * (np.sinh(6*N_p))
            + ap8 * (np.cos(8*E_p)) * (np.sinh(8*N_p))
            + ap10 * (np.cos(10*E_p)) * (np.sinh(10*N_p))
            + ap12 * (np.cos(12*E_p)) * (np.sinh(12*N_p)))
```

```
# solve for X and Y with adjustment for UTM scale factor
X = k0 * A * Ncoordonate
Y = k0 * A * Ecoordonate

# The following are not needed to complete the coordinate
# system conversion but are provided for completeness

# # compute p and q values
# p_p=1.+2*ap2*(np.cos(2*E_p))*(np.cosh(2*N_p)) + \
#     4*ap4*(np.cos(4*E_p))*(np.cosh(4*N_p)) + \
#     6*ap6*(np.cos(6*E_p))*(np.cosh(6*N_p)) + \
#     8*ap8*(np.cos(8*E_p))*(np.cosh(8*N_p)) + \
#     10*ap10*(np.cos(10*E_p))*(np.cosh(10*N_p)) + \
#     12*ap12*(np.cos(12*E_p))*(np.cosh(12*N_p))


# q_p=2*ap2*(np.sin(2*E_p))*(np.sinh(2*N_p)) + \
#     4*ap4*(np.sin(4*E_p))*(np.sinh(4*N_p)) + \
#     6*ap6*(np.sin(6*E_p))*(np.sinh(6*N_p)) + \
#     8*ap8*(np.sin(8*E_p))*(np.sinh(8*N_p)) + \
#     10*ap10*(np.sin(10*E_p))*(np.sinh(10*N_p)) + \
#     12*ap12*(np.sin(12*E_p))*(np.sinh(12*N_p))

# # calculate convergence
# gamma_p=np.arctan((T_p*tanlam)/(((1+T_p*T_p)**.5)))
# gamma_pp=np.arctan2(q_p,p_p)
# gamma=gamma_p+gamma_pp

# # Calculate scale factor
# sinphi=np.sin(phi)
# k_p=(1-e*e*sinphi*sinphi)**.5*(1+tanphi*tanphi)**.5 / \
# (T_p*T_p+coslam*coslam)**.5
# k_pp=A/a*np.sqrt(p_p*p_p+q_p*q_p)
# k=k0*k_p*k_pp

# shift x/y to false origins
E=X+FalseEasting

if np.all(h == 'S'):
   N = Y + FalseNorthing  # y in southern hemisphere relative false
                          # northing
else:
   N = Y

if trunc_val == 0:
   E = trunc(E, trunc_val)  # x relative to false easting
   N = trunc(N, trunc_val)

elif trunc_val != None:
   E = trunc(E, trunc_val)  # x relative to false easting
   N = trunc(N, trunc_val)

   # add zero padding
   E = "{:06.0f}".format(E)
   N = "{:07.0f}".format(N)
```

```
    return int(zone), h, E, N

# ----------------------------------------------------------------------
# SECTION 2.2: Convert to planetocentric Lat and Lon from LTM


def toLatLon(E, N, zone, h, process_errors=True):
    """Converts LTM easting and northing coordinates to Planetocentric
       LatLon degree values. Eastings, Northings, Zone, and hemisphere
       are needed inputs to complete the inverse conversion. Zone and
       hemisphere must be specified; however, this is to the user to
       specify; This allows forced processing of an adjacent zone.

    Parameters:
    zone           LTM zone              (int,float) scaler
    h              hemisphere            (str) unitless
    E              Eastings              (float,str) meters
    N              Northings             (float,str) meters
    process_errors  Processing flag      (logical): boolean

    Returns:
    lam            Longitude             (float): degrees
    phi            Latitude              (float): degrees

    Raises:
    ValueError: If coordinates are not within the proper LTM range of,
            125,000m ≤ E < 375,000m
            0m ≤ N < 2,500,000m
            The program terminates if latitude is not in
            coverage area.
            """

    if isinstance(E, str):
        E = float(E)
        N = float(N)

    if process_errors:
        if E > 375000:
            raise ValueError("Operation aborted: Easting less than 375km.")
        elif E < 125000:
            raise ValueError("Operation aborted: Easting less than 125km.")
        elif N > 2500000:
            raise ValueError("Operation aborted: Northing exceeds 2,500km")
        elif N < 0:
            raise ValueError("Operation aborted: Northing negative")

    # subtract false origins for the easting and northing values
    X = E - FalseEasting

    if h == 'S':
        Y = N - FalseNorthing  # if in the southern hemisphere
    else:
        Y = N

    # Calculate Unscaled transverse Mercator ratios
```

```
N = X / (k0*A)
E = Y / (k0*A)

# Compute the Gauss-Schreiber ratios using Clenshaw summation
E_p = (E
   - bt2 * np.sin(2*E) * np.cosh(2*N)
   - bt4 * np.sin(4*E) * np.cosh(4*N)
   - bt6 * np.sin(6*E) * np.cosh(6*N)
   - bt8 * np.sin(8*E) * np.cosh(8*N)
   - bt10 * np.sin(10*E) * np.cosh(10*N)
   - bt12 * np.sin(12*E) * np.cosh(12*N))

N_p = (N
   - bt2 * (np.cos(2*E)) * (np.sinh(2*N))
   - bt4 * (np.cos(4*E)) * (np.sinh(4*N))
   - bt6 * (np.cos(6*E)) * (np.sinh(6*N))
   - bt8 * (np.cos(8*E)) * (np.sinh(8*N))
   - bt10 * (np.cos(10*E)) * (np.sinh(10*N))
   - bt12 * (np.cos(12*E)) * (np.sinh(12*N)))

sinhN_p = np.sinh(N_p)
sinE_p = np.sin(E_p)
cosE_p = np.cos(E_p)

# Use Newton-Raphson iteration values to solve for Latitude
T_p = sinE_p / np.sqrt(sinhN_p*sinhN_p + cosE_p*cosE_p)

dTi = 1.
Ti = np.zeros(T_p.shape)
Ti = np.array(T_p)  # deeps copy of array!!

# For i in range(0,Ti.shape[0])
i = 0

while np.abs(dTi) > 1E-12:
   sigmai = np.sinh(e * np.arctanh(e*Ti / np.sqrt(1 + Ti*Ti)))

   Ti_p = Ti * np.sqrt(1 + sigmai*sigmai) - sigmai*np.sqrt(1 + Ti*Ti)

   dTi = ((T_p - Ti_p) / np.sqrt(1 + Ti_p*Ti_p)
         * (1 + (1 - e*e)*Ti*Ti) / ((1 - e*e)*np.sqrt(1 + Ti*Ti)))
   Ti += dTi

   # break
   if i > 50:
      raise ValueError("Operation aborted: latitude iteration "
                  "did not converge")
   i += 1

T = Ti
phi = np.arctan(T)

lam = np.arctan2(sinhN_p, cosE_p)

# equations are provided for completeness but not required to
# complete the conversion
```

```
    # # convergence: Karney 2011 Eq 26, 27
    # p=1.-2*bt2*(np.cos(2*E))*(np.cosh(2*N)) - \
    #      4*bt4*(np.cos(4*E))*(np.cosh(4*N)) - \
    #      6*bt6*(np.cos(6*E))*(np.cosh(6*N)) - \
    #      8*bt8*(np.cos(8*E))*(np.cosh(8*N)) - \
    #      10*bt10*(np.cos(10*E))*(np.cosh(10*N)) - \
    #      12*bt12*(np.cos(12*E))*(np.cosh(12*N))


    # q=2*bt2*(np.sin(2*E))*(np.sinh(2*N)) + \
    #   4*bt4*(np.sin(4*E))*(np.sinh(4*N)) + \
    #   6*bt6*(np.sin(6*E))*(np.sinh(6*N)) + \
    #   8*bt8*(np.sin(8*E))*(np.sinh(8*N)) + \
    #   10*bt10*(np.sin(10*E))*(np.sinh(10*N)) + \
    #   12*bt12*(np.sin(12*E))*(np.sinh(12*N))

    # gamma_p=np.arctan(np.tan(E_p)*np.tanh(N_p))
    # gamma_pp=np.arctan2(q,p)
    # gamma=gamma_p+gamma_pp

    # # scale: Karney 2011 Eq 28
    # sinphi=np.sin(phi)
    # k_p=(1.-e*e*sinphi*sinphi)**.5*(1 + T*T)**.5\
    #     *(sinhN_p*sinhN_p+ cosE_p*cosE_p)**.5
    # k_pp=A/a/(p*p + q*q)**.5
    # k=k0*k_p*k_pp

    # redetermine the central meridian from the zone provided
    lam0 = np.deg2rad((zone - 1) * (ZoneWidth*2) - 180 + ZoneWidth)
    lam += lam0  # adjust longitude from zonal to global coordinates

    return np.rad2deg(lam), np.rad2deg(phi)  # convert back to degrees

# ----------------------------------------------------------------------
# SECTION 2.3: Calculate meridional radius of curvature


def meridional_radius_of_curvature(phi, phi1, e):
    """Calculates curvature needed to support an ellipsoid for a polar
   stereographic map projection.
    Parameters:
    phi     latitude (float): degrees
    phi1    projection longitudinal reference (float): degrees
    e       ellipsoid eccentricity (float): unitless

    Returns:
    t     (float): Unitless, curvature"""
    pi2 = np.pi/2.
    pi4 = pi2/2.

    if phi1 == np.deg2rad(-90.):  # sign is flipped
        t = np.tan(pi4 - (-phi/2)) / (((1 - e*np.sin(-phi)) / (1 + e*np.sin(-phi)))**(e/2))
    else:
        t = np.tan(pi4 - (phi/2)) / (((1 - e*np.sin(phi)) / (1 + e*np.sin(phi)))**(e/2))
    return t
```

```
# ---------------------------------------------------------------------
# SECTION 2.4: Calculate distance from pole to latitude parallel


def ellipsoidal_polar_arc_distance(a, k_0, e, t):
    """Calculates the distance from the pole to the points latitude
    Parameters:
    a       reference surface radius (float): meters
    k0      projection central scale factor (float): unitless
    e       ellipsoid eccentricity (float): unitless
    t       ellipsoid eccentricity (float): unitless

    Returns:
    Rho     (float): meters, Distance from pole to points"""
    pe = (1 + e)
    me = (1 - e)
    ang = np.sqrt((pe**pe) * (me**me))
    rho = 2*a*k_0*t/ang
    return rho

# ---------------------------------------------------------------------
# SECTION 2.5: Calculate polar stereographic ellipsoidal X coordinate


def ellipsoidal_stereographic_map_x(A, phiX, phiX1, lam, lam0, rho=0):
    """Determine the X coordinate for a stereographic map projection.
    Reduced equations are provided for the polar stereographic maps
    as was done in the reference material. See Snyder (1987) for more
    information.

    map projection from a ellipsoid reference surface.
    Parameters:
    A         Gaussian Radius of curvature (float): meters
    phiX      Conformal latitude (float): degrees
    phiX1     Conformal Projection origin latitude (float): degrees
    lam       longitude (float): degrees
    lam0      Projection origin latitude (float): degrees
    rho       Radius from center for projection at {phiX1,lam0}
              (float): unitless

    Returns:
    map_X float: grid Coordinate (float) meters"""

    if phiX1 == np.deg2rad(90.):
        map_x = rho*np.sin(lam - lam0)


    elif phiX1 == np.deg2rad(-90):
        map_x = rho*np.sin(lam - lam0)


    else:
        map_x = A*np.cos(phiX)*np.sin(lam - lam0)

    return map_x
```

```
# ----------------------------------------------------------------------
# SECTION 2.6: Calculate polar stereographic ellipsoidal Y coordinate


def ellipsoidal_stereographic_map_y(A, phiX, phiX1, lam, lam0, rho=0):
    """Determines the Y coordinate for a stereographic map projection.
    Reduced equations are provided for the polar stereographic maps
    as was done in the reference material. See Snyder (1987) for more
    information.

    map projection from a ellipsoidal reference surface.
    Parameters:
    A         Gaussian Radius of curvature (float): meters
    phiX       Conformal latitude (float): degrees
    phiX1      Conformal Projection origin latitude (float): degrees
    lam        longitude (float): degrees
    lam0       Projection origin latitude (float): degrees
    rho        Radius from center for projection at {phiX1,lam0}
               (float): unitless

    Returns:
    map_Y float: grid Coordinate (float) meters"""
    # Map Y coordinate

    if phiX1 == 0.:  # equation for equatorial center
        y = A*np.sin(phiX)

    elif phiX1 == np.deg2rad(90.):
        map_y = -rho*np.cos(lam - lam0)

    elif phiX1 == np.deg2rad(-90):
        map_y = rho*np.cos(-lam - lam0)

    else:  # oblique aspects
        ang = np.cos(phiX1)*np.sin(phiX) - np.sin(phiX1)*np.cos(phiX)*np.cos(lam - lam0)
        map_y = A*ang
    return map_y

# ----------------------------------------------------------------------
# SECTION 2.7: Calculate polar stereographic spherical scale error


def polar_stereographic_spherical_scale_err(phi, phi_1, lam, lam0, k_0):
    #scale error in azimuthal projection
    """Determines the map point scale error for a polar stereographic
    map projection from a spheroidal reference surface.
    Parameters:
    phi         latitude (float): degrees
    phi_1       Projection origin latitude (float): degrees
    lam         longitude (float): degrees
    lam0        Projection origin latitude (float): degrees
    k0          Central scale factor, located at {phi_1,lam0}
                (float): unitless

    Returns:
```

```
    k float,int: point scale error.""""

    if phi_1 == -90:  # South polar stereographic specific function
        k = (2*k_0) / (1 - np.sin(phi))

    elif phi_1 == 90:  # North pole specific function
        k = (2*k_0) / (1 + np.sin(phi))

    # Process all other areas (function can be used for oblique stereographic)
    else:
        k = (2*k_0) / (1 + np.sin(phi_1)*np.sin(phi) + np.cos(phi_1)*np.cos(phi)*np.cos(lam - lam0))
    return k

# ----------------------------------------------------------------------
# SECTION 2.8: Calculate polar stereographic spherical X coordinate


def spherical_stereographic_map_x(phi, phi_1, lam, lam0, R, k):
    """Determines the X coordinate for a stereographic map projection.
    Reduced equations are provided for the polar stereographic maps
    as was done in the reference material. See Snyder (1987) for more
    information.

    map projection from a spheroidal reference surface.
    Parameters:
    phi         latitude (float): degrees
    phi_1       Projection origin latitude (float): degrees
    lam         longitude (float): degrees
    lam0        Projection origin latitude (float): degrees
    R           reference surface radius (float): meters
    k           point scale factor, located at {phi_1,lam0}
                (float): unitless

    Returns:
    map_X float: grid Coordinate."""

    if phi_1 == -90: # south polar stereographic specific function
        map_x = 2*R*k0_polar*np.tan(np.pi/4 + phi/2)*np.sin(lam - lam0)

    elif phi_1==90: # north pole specific function
        map_x = 2*R*k0_polar*np.tan(np.pi/4 - phi/2)*np.sin(lam - lam0)

    else: # Process all other areas
        map_x = R*k*np.cos(phi)*np.sin(lam - lam0)
    return map_x

# ----------------------------------------------------------------------
# SECTION 2.9: Calculate polar stereographic spherical Y coordinate


def spherical_stereographic_map_y(phi, phi_1, lam, lam0, R, k):  # Map Y coordinate
    """Determines the Y coordinate for a stereographic map projection.
    Reduced equations are provided for the polar stereographic maps
    as was done in the reference material. See Snyder (1987) for more
    information.
```

```
    map projection from a spheroidal reference surface.
    Parameters:
    phi         latitude (float): degrees
    phi_1       Projection origin latitude (float): degrees
    lam         longitude (float): degrees
    lam0        Projection origin latitude (float): degrees
    R           reference surface radius (float): meters
    k           point scale factor, located at {phi_1,lam0}
                (float): unitless

    Returns:
    map_Y float: grid Coordinate."""

    if phi_1 == -90: # south polar stereographic specific function
        map_y = 2*R*k0_polar*np.tan(np.pi/4 + phi/2)*np.cos(lam - lam0)

    elif phi_1 == 90: # north pole specific function
        map_y = 2*R*k0_polar*np.tan(np.pi/4 - phi/2)*np.cos(lam - lam0)

    else: # Process all other areas
        ang = np.cos(phi_1)*np.sin(phi) - np.sin(phi_1)*np.cos(phi)\
            * np.cos(lam - lam0)
        map_y = R*k*ang

    return map_y

# ----------------------------------------------------------------------
# SECTION 2.10: Convert to LPS from planetocentric Lat and Lon


def toLPS(lam, phi, trunc_val=1, eqs="Spherical", process_errors=True):
    """Converts Planetocentric LatLon degree values to LPS Easting
       and Northing Coordinates. Latitude and Longitude are the only
       Inputs required. The LPS zone/ hemisphere does not have to be
       specified as a simple test on the coordinates can determine
       the hemisphere. An option is available to truncate the
       coordinates. 1m is the default needed for LPS coordinates
       specifically; however, there will be round off with inverse
       conversions or additional conversions to other coordinates.
       For conversion to a grid systems value, set to 0 for no
       truncation.

    Parameters:
    lam             Longitude             (float): degrees
    phi             Latitude              (float): degrees
    equations       ellipsoidal or spherical (str): unitless
    trunc_val       Coordinate Precision    (int,float): degrees
    process_errors  Processing flag         (logical): boolean

    Returns:
    h               LTM zone/hemisphere     (str) unitless
    E               Eastings              (float,str) meters
    N               Northings             (float,str) meters

    Raises:
    ValueError: If coordinates are not within a latitude range
```

```
            80 to 90 if h==”N” or if
            -90 to -80 if h==”S” or if,
            longitude is not in the -180-180 range.
            Program will terminate if no projection method is
            Specified”””

    if process_errors:

        if phi > 90:
            raise ValueError(“Operation aborted: Latitude exceeds 90°.”)
        elif phi < -90:
            raise ValueError(“Operation aborted: Latitude less than -90°.”)
        elif lam > 180:
            raise ValueError(“Operation aborted: Longitude exceeds 180° “
                             “ Try converting from 0°-360° to “
                             “-180°-180° longitude range.”)

        elif lam < -180:
            raise ValueError(“Operation aborted: Longitude is less than 180°”)
        elif (phi < 80 and phi > -80):
            raise ValueError(“Operation aborted: Latitude outside of “
                             “LPS Polar projection areas”)

    # force -180 to 180. Fixes East West Naming differences in LGRS polar
    # geometrically this is the same point.

    if lam == -180:
        lam = 180

    # Determine the LPS Zone/hemisphere for processing
    if phi >= 80.:
        h = ‘N’                 # north hemisphere
        phi1 = np.deg2rad(90.)  # projection origin latitude

    elif phi <= -80.:
        h = ‘S’                 # south hemisphere
        phi1 = np.deg2rad(-90.)  # projection origin latitude

    else:
        raise ValueError(“Hemisphere could not be determined”)

    lam0 = np.deg2rad(0.)  # projection origin longitude

    # Convert degrees to radians
    phi = np.deg2rad(phi)
    lam = np.deg2rad(lam)

    # Snyder (1987) equations for converting to and from
    # polar stereographic. Two methods were provided from Snyder.
    # both are provided here to support a more complex lunar shape
    # if needed in the future.
    if eqs == “Spherical”:    # Snyder (1987) spherical equations
        k = polar_stereographic_spherical_scale_err(phi, phi1, lam, lam0, k0_polar)  # calculate scale factor
        StereoX = spherical_stereographic_map_x(phi, phi1, lam, lam0, a, k)  # determine grid X value
        StereoY = spherical_stereographic_map_y(phi, phi1, lam, lam0, a, k)  # determine grid Y value
```

```
    elif eqs == “Ellipsoidal”:    # Snyder (1987) ellipsoidal equations
        t = meridional_radius_of_curvature(phi, phi1, e)  # calculate radial curvature
        rho = ellipsoidal_polar_arc_distance(a, k0_polar, e, t)  # calculate distance from center to point
        StereoX = ellipsoidal_stereographic_map_x(a, phi, phi1, lam, lam0, rho)  # determine grid X value
        StereoY = ellipsoidal_stereographic_map_y(a, phi, phi1, lam, lam0, rho)  # determine grid Y value

    else:
        raise ValueError(“Operation aborted: Conversion method not “
                         “specified”)


    # Add false Eastings and Northings
    StereoX += FalseEasting_polar
    StereoY += FalseNorthing_polar

    if trunc_val == 0:
        StereoX = trunc(StereoX, trunc_val)  # x relative to false easting
        StereoY = trunc(StereoY, trunc_val)

    elif trunc_val != None:
        StereoX = trunc(StereoX, trunc_val)  # x relative to false easting
        StereoY = trunc(StereoY, trunc_val)

        # add zero padding
        StereoX=”{:06.0f}”.format(StereoX)
        StereoY=”{:06.0f}”.format(StereoY)
    # zone is set to an empty value as it is not needed for LPS
    # zone=None

    # Return LPS coordinates to the user.
    return h, StereoX, StereoY

# ----------------------------------------------------------------------
# SECTION 2.11: Inverse calculate meridional radius of curvature


def inv_meridional_radius_of_curvature(rho, e, a, k0, phi1):
    “””Calculates curvature needed to support an ellipsoid for a polar
    stereographic map projection
    Parameters:
    rho     distance between pole and point (float): meters
    e       eccentricity (float): unitless
    a       Reference surface semi major radius (float): meters
    k0      map projection central scale factor
    phi1    projection latitude reference (float): degrees

    Returns:
    t     (float): Unitless, curvature”””

    pe = (1 + e)
    me = (1 - e)

    if phi1 == np.deg2rad(-90) or phi1 == np.deg2rad(90):
        t = rho * np.sqrt((pe**pe) * (me**me)) / (2*a*k0)

    else:
```

```
        # requires additional functions beyond the scope of LPS
        raise ValueError("Operation aborted: ellipsoidal curvature "
                         "Not be determined.")
    return t

# ----------------------------------------------------------------------
# SECTION 2.12: Calculate conformal latitude


def ellipsoidal_stereographic_conformal_latitude(phi, e, phi1=0, t=None):
    """Calculates conformal latitude on the ellipsoid from a polar
    stereographic map projection points

    Parameters:
    phi     latitude (float): degrees
    e       eccentricity (float): unitless
    phi1    projection latitude reference (float): degrees

    Function has two states, one for polar and one for oblique
    orientations.

    Returns:
    X     conformal latitude (float): Unitless"""
    pi2 = np.pi/2.
    pi4 = pi2/2.

    # polar regions phi not needed
    if phi1 == np.deg2rad(90) or phi1 == np.deg2rad(-90):
        X = pi2 - 2*np.arctan(t)

    # equatorial projection regions, phi needed but t is.
    else:
        ang = np.tan(pi4 + (phi/2))*(((1 - e*np.sin(phi))/(1 + e*np.sin(phi)))**(e/2))
        X = 2*np.arctan(ang) - pi2
    return X

# ----------------------------------------------------------------------
# SECTION 2.13: Recover Latitude on an ellipsoid without iteration


def ellipsoidal_stereographic_latitude(phiX, e, phi1X):
    """Recovers latitude on the ellipsoid from a polar
    stereographic map projection points. Program uses an approximation
    described in Snyder (1987), to avoid iteration.

    Parameters:
    phiX    conformal latitude (float): degrees
    e       eccentricity (float): unitless
    phi1X   Conformal projection latitude reference (float): degrees

    Returns:
    phi     latitude (float): degrees"""

    phi=(phiX
        + C2*np.sin(2*phiX)
        + C4*np.sin(4*phiX)
```

```
        + C6*np.sin(6*phiX)
        + C8*np.sin(8*phiX))

    # assign correct hemispheric location based on map projection
    if phi1X == np.deg2rad(-90):
        return phi * -1
    else:
        return phi

# ---------------------------------------------------------------------
# SECTION 2.14: Recover Longitude on an ellipsoid


def ellipsoidal_stereographic_longitude(x, y, C_e, rho, phi1X, lam0):
    """Recovers longitude on the ellipsoid from a polar
    stereographic map projection points.

    Parameters:
    x       grid coordinate (float): meters
    y       grid coordinate (float): meters
    C_e     Simplified angular distance (float): see Snyder (1987)
            Not needed for polar areas.
    rho     Radius from center for projection at {phiX1,lam0}
             (float): unitless
    phi1X   Conformal projection latitude reference (float): degrees
    lam0    projection longitudinal reference (float): degrees

    Returns:
    lam     longitude (float): degrees"""

    if phi1X == np.deg2rad(-90):
        lam = -lam0+np.arctan2(x, -y)
        lam = np.deg2rad(planetocentric_lon_degrees(
                            np.rad2deg(-lam + np.deg2rad(180))))

    elif phi1X==np.deg2rad(90): # this needs to be checked
        lam = lam0 + np.arctan2(x, y)
        lam = np.deg2rad(planetocentric_lon_degrees(
                            np.rad2deg(-lam - np.deg2rad(180))))
    else:
        ang = (x*np.sin(C_e))/\
        (rho*np.cos(phi1X)*np.cos(C_e) - y*np.sin(phi1X)*np.sin(C_e))
        lam = lam0 + np.arctan(ang)

    return lam

# ---------------------------------------------------------------------
# SECTION 2.15: Calculate distance from projection pole to parallel


def Euclidean_distance2D(x, y):
    """Calculates the distance from the pole to the points latitude
    Parameters:
    X       Polar stereographic X coordinate (float): meters
    Y       Polar stereographic Y coordinate (float): meters
```

```
    As coordinates are Cartesian, Euclidean distance works here.

    Returns:
    Rho     (float): meters, Distance from pole to points"""

    return (x**2 + y**2)**.5

# ----------------------------------------------------------------------
# SECTION 2.16: Calculate great circle distance from pole to parallel


def polar_stereographic_great_cicle_arc(rho, R, k0):
    """Calculates the distance from the pole to the points latitude
    Parameters:
    Rho     (float): meters, Distance from pole to points
    R       (float): reference surface radius
    k0      Central scale factor, located at {phi_1,lam0}
              (float): unitless
    As coordinates are Cartesian Euclidean distance works here.

    Returns:
    c       (float) great circle arc distance on a sphere"""

    return 2 * np.arctan2(rho, (2*R*k0))

# ----------------------------------------------------------------------
# SECTION 2.17: Recover latitude on a sphere


def spherical_polar_latitude(c, phi_1, y, rho):
    """Determines the longitude from a stereographic map projection.
    Coordinates must be in polar stereographic topocentric coordinates
    See Snyder (1987) for more information.

    map projection inverse conversion for a spherical reference surface.
    Parameters:
    c          great circle distance (float): meters
    phi_1      Projection origin latitude (float): degrees
    y          grid coordinate (float): meters
    rho        distance from the pole to the points latitude {phiX1,lam0}
               (float): unitless

    Returns:
    phi  latitude  (float) degrees"""

    if rho == 0. and c == 0.:
        phi=phi_1
    else:
        ang = ((np.cos(c)*np.sin(phi_1)) + ((y*np.sin(c)*np.cos(phi_1))/rho))
        phi = np.arcsin(ang)

    # this is a logical check to see if the point is the same as the
    # map projection origin. If so then we assign the origin point
    # explicitly
    if np.isnan(phi):
        phi = phi_1
```

```
    return phi

# ----------------------------------------------------------------------
# SECTION 2.18: Recover longitude on a sphere


def spherical_polar_longitude(x, c, rho, phi_1, y, lam0):
    """Determines the longitude from a stereographic map projection.
    Coordinates must be in polar stereographic topocentric coordinates
    See Snyder (1987) for more information.

    map projection inverse conversion for a spherical reference surface.
    Parameters:
    x          grid Coordinate (float): meters
    c          great circle distance (float): meters
    rho        distance from the pole to the points latitude {phiX1,lam0}
               (float): unitless
    phi_1      Projection origin latitude (float): degrees
    y          grid Coordinate (float): meters
    lam0       Projection origin longitude (float): degrees

    Returns:
    lam  longitude  (float) degrees"""

    if phi_1 == np.deg2rad(90.): # process north pole
        lam = lam0 + np.arctan2(x, (-1*y))

    elif phi_1 == np.deg2rad(-90.):# process south pole
        lam = lam0 + np.arctan2(x,y)

    else: # process all other areas
        ang = np.arctan2((x*np.sin(c)),
            (rho*np.cos(phi_1)*np.cos(c) - y*np.sin(phi_1)*np.sin(c)))
        lam = lam0 + ang

    return lam

# ----------------------------------------------------------------------
# SECTION 2.19: Convert to planetocentric Lat and Lon from LPS


def toLatLon_Polar(E, N, h, eqs="Spherical", process_errors=True):
    """Converts LPS hemisphere, Easting, Northing Coordinates to
       Planetocentric LatLon degree values. Eastings, Northings, Zone,
       and hemisphere. Zone is not needed as the hemisphere is used.

    Parameters:
    zone           LTM zone              (int,float) scaler
    h              hemisphere            (str) unitless
    E              Eastings              (float,str) meters
    N              Northings             (float,str) meters
    process_errors  Processing flag      (logical): boolean

    Returns:
    lon            Longitude             (float): degrees
```

```
lat          Latitude          (float): degrees

Raises:
ValueError: If coordinates are not within a proper LPS range of,
        197,000m ≤ E < 805000m
        197,000m ≤ N < 805000m
        Program terminates if latitude does not converge
        “””

if isinstance(E,str):
   E=float(E)
   N=float(N)

if process_errors:
   if E > 805000:
      raise ValueError(“Operation aborted: Easting outside of value range.”)
   elif E < 197000:
      raise ValueError(“Operation aborted: Easting outside of value range.”)
   elif N > 805000:
      raise ValueError(“Operation aborted: Northing outside of value range.”)
   elif N < 197000:
      raise ValueError(“Operation aborted: Northing outside of value range”)

# Determine the LPS Zone/hemisphere for processing, hemisphere is a
# required argument for processing so we use it here
if h == ‘N’: # north polar
   phi1=np.deg2rad(90) # projection reference latitude
elif h == ‘S’: # south polar
   phi1 = np.deg2rad(-90)

lam0 = np.deg2rad(0.) # projection reference longitude

# remove false origins
X = E - FalseEasting_polar
Y = N - FalseNorthing_polar

# Snyder (1987) Spherical Equations
if eqs == “Spherical”:
   Rho = Euclidean_distance2D(X, Y) # determine distance from pole to points
   C = polar_stereographic_great_cicle_arc(Rho, a, k0_polar) # calculate arc distance
   phi = spherical_polar_latitude(C, phi1, Y, Rho) # determine latitude
   lam = spherical_polar_longitude(X, C, Rho, phi1, Y, lam0) # determine longitude

# Snyder (1987) ellipsoidal equations
elif eqs == “Ellipsoidal”:
   rho = Euclidean_distance2D(X, Y) # determine distance from pole to points
   t = inv_meridional_radius_of_curvature(rho, e, a, k0_polar, phi1)
   phiX = ellipsoidal_stereographic_conformal_latitude(None, e, phi1, t)
   phi = ellipsoidal_stereographic_latitude(phiX, e, phi1)
   lam = ellipsoidal_stereographic_longitude(X, Y, 0, rho, phi1, lam0)

else:
   raise ValueError(“Operation aborted: Conversion method not “
                    “specified”)

if lam == -np.pi:
```

```
        lam = np.pi

    return np.rad2deg(lam), np.rad2deg(phi)

# ----------------------------------------------------------------------
# SECTION 2.20: Convert to LGRS from LTM


def toLGRS(X, Y, lonBand, h, trunc_val=1 ,process_errors=True):
    """Converts LTM Easting and Northing Coordinates to LGRS coordinate
    of longitude band (LTM zone),latitude band, 25km easting grid letter,
    25km northing grid letter, Easting in 25km grid zone, and Northing
    in 25km grid zone. All LTM coordinates are required for proper
    output.
    If coordinate is truncated then the coordinate is zero padded
    to the proper string length. if not truncated, the final coordinate
    is left as a floating point.

    Parameters:
    X              LTM Eastings           (float,int) meters
    Y              LTM Northings          (float,int) meters
    lonBand        LTM zone               (int,float) scaler
    h              hemisphere             (str) unitless
    trunc_val      precision              (int,float): meters
    process_errors  Processing flag       (logical): boolean

    Returns:
    lonBand        LGRS zone (LTM zone)    (float) scaler
    latBand        C-X 8° latitude reference (str)
    e25k           25km grid zone easting letter (str)
    n25k           25km grid zone northing letter (str)
    E              25km grid zone easting (str,float) meters
    N              25km grid zone northing (str,float) meters

    Raises:
    ValueError: If coordinates are not within a the proper LTM range of,
            125,000m ≤ E < 375,000m
            0m ≤ N < 2,500,000m
            or has the correct zone
            """
    if process_errors:
        if X > 375000:
            raise ValueError("Operation aborted: Easting less than 375km.")
        elif X < 125000:
            raise ValueError("Operation aborted: Easting less than 125km.")
        elif Y > 2500000:
            raise ValueError("Operation aborted: Northing exceeds 2,500km")
        elif Y < 0:
            raise ValueError("Operation aborted: Northing negative")
        elif lonBand < 1:
            raise ValueError("Operation aborted: Incorrect LTM zone formatting")
        elif lonBand > 45:
            raise ValueError("Operation aborted: Incorrect LTM zone formatting")
    # letter set for latitude zones and 25km grid zones
    # determined above.
```

```
    # number of unique letter sets for LGRS
    cols=1
    rows=3

    # global values | false easting and northing values must be applied

    # convert LTM to LatLon to get latitude to determine the correct band
    _,lat=toLatLon(X,Y,lonBand,h,'Lunar')

    # grid zones are 8°/10° tall, at 0° N is 10th band: only works w/ 8°
    latBand = latBands[int(np.floor(lat/(2*ZoneWidth) + len(latBands)/2))]

    # 25km zones
    col=int(np.floor(X/25E3)) # determine Easting letter index
    row = int(np.floor(Y/25E3) % 20) # determine Northing letter index

    # assign Easting letter by index. Five is subtracted to center the
    # letter reference
    e25k = e25kLetters[int((lonBand-1)%cols)][col-5]

    # assign Northing letter by index
    # rows in zones are A-V, F-E, or L-K.
    n25k = n25kLetters[int((lonBand-1)%rows)][row]

    # truncate easting/northing to within 25km grid square
    E = X % 25E3
    N = Y % 25E3

    # truncate to format final output
    if trunc_val != None:
        E = trunc(E,trunc_val,process_errors=True)
        N = trunc(N,trunc_val,process_errors=True)

        # add zero padding to value. Convert to strings to retain vals
        if trunc_val != 0:
            E = "{:05.0f}".format(E)
            N = "{:05.0f}".format(N)

    return lonBand, latBand, e25k, n25k, E, N

# ----------------------------------------------------------------------
# SECTION 2.21: Convert to LGRS from LPS


def toLGRS_polar(E, N, h, trunc_val=1, process_errors=True):
    """Converts LTM Easting and Northing Coordinates to LGRS coordinate
    of longitude band (LTM zone),latitude band, 25km easting grid letter,
    25k northing grid letter, Easting in 25km grid zone, and Northing
    in 25km grid zone on polar regions. All LTM coordinates are required
    for proper output.
    If coordinate is truncated then the coordinate is zero padded
    to the proper string length. if not truncated, the final coordinate
    is left as a floating point.

    Parameters:
    X           LPS Eastings          (float,str) meters
```

```
Y              LPS Northings          (float,str) meters
h              hemisphere             (str) unitless
trunc_val      precision              (int,float): meters
process_errors  Processing flag       (logical): boolean

Returns:
lonBand        LGRS zone A,B,Y,Z    (str) scaler
e25k           25km grid zone easting letter (str)
n25k           25km grid zone northing letter (str)
E              25km grid zone easting (str,float) meters
N              25km grid zone northing (str,float) meters

Raises:
ValueError: If coordinates are not within a proper LPS range of,
        197,000m ≤ E < 805000m
        197,000m ≤ N < 805000m
        Program terminates if coordinates are in the wrong zone
        “””
if isinstance(E,str):
   E=float(E)
   N=float(N)

if process_errors:
   if E > 805000:
      raise ValueError(“Operation aborted: Easting outside of value range.”)
   elif E < 197000:
      raise ValueError(“Operation aborted: Easting outside of value range.”)
   elif N > 805000:
      raise ValueError(“Operation aborted: Northing outside of value range.”)
   elif N < 197000:
      raise ValueError(“Operation aborted: Northing outside of value range”)
   elif h not in [“S”, “N”]:
      raise ValueError(“Operation aborted: Wrong hemisphere provided”)

# lat_band
# there is no latitude band on the poles

# lon_Band
# determine the longitudinal LGRS zone with logical checks
if h == ‘S’:
   if E < FalseEasting_polar :
      lonBand = ‘A’
   elif E >= FalseEasting_polar:
      lonBand = ‘B’
elif h == ‘N’:
   if E < FalseEasting_polar:
      lonBand = ‘Y’
   elif E >= FalseEasting_polar:
      lonBand = ‘Z’

# letter set for 25km grid zones determined above.
# e25kLetters_polar
# n25kLetters_polar

# make coordinates relative to false northing or easting
X = E - FalseEasting_polar
```

```
    Y = N - FalseNorthing_polar

    # determine center index value
    centX = (len(e25kLetters_polar) - 1)//2 # this is index 13
    centY = (len(n25kLetters_polar) - 1)//2 #

    # shift increases from A to the East. Decrements from Z to west
    if lonBand == 'A' or lonBand == 'Y':
        col = int((len(n25kLetters_polar) - 1) - (np.floor(np.abs(X)/25E3)))
        # E-=25E3
    else:
        col = int((np.floor(X/25E3)))
    e25k = e25kLetters_polar[col]

    # add half the letter width to the index to correct position
    # done b/c everything is relative to the center
    row = int(np.floor(Y/25E3) + centY) + 1

    # added extended range characters
    if row < 0:
        n25k = '-'
    elif row > len(n25kLetters_polar) - 1:
        n25k = '+'
    else:
        n25k = n25kLetters_polar[row]

    # truncate easting/northing to within 25km grid square

    if lonBand == 'A' or lonBand == 'Y':
        E = 25E3-np.abs(X) % 25E3
    else:
        E = np.abs(X) % 25E3
    N = Y % 25E3

    # truncate to format final output
    if trunc_val != None:
        E = trunc(E, trunc_val, process_errors=True)
        N = trunc(N, trunc_val, process_errors=True)

    # add zero padding to value. Convert to strings to retain vals
        if trunc_val != 0:
            E = "{:05.0f}".format(E)
            N = "{:05.0f}".format(N)

    return lonBand, e25k, n25k, E, N

# ----------------------------------------------------------------------
# SECTION 2.22: Convert to LTM from LGRS


def LGRStoLTM(latBand, lonBand, e25k, n25k, E, N, process_errors=True):
    """Converts LGRS coordinate of longitude band (LTM zone),latitude
    band, 25km easting grid letter, 25k northing grid letter, Easting
    in 25km grid zone , and Northing in 25km grid zone to LTM Easting
    and Northing Coordinates. All LGRS coordinates are required for proper
    output.
```

```
If coordinate is truncated previously there will be some precision
loss when converting back to LatLon.
Conversion requires 2LTM function.

Parameters:

lonBand       LGRS zone (LTM zone)    (float) scaler
latBand       C-X 8° latitude reference (str)
e25k          25km grid zone easting letter (str)
n25k          25km grid zone northing letter (str)
E             25km grid zone easting (str,float) meters
N             25km grid zone northing (str,float) meters

Returns:
lonBand       LTM zone              (int,float) scaler
h             hemisphere            (str) unitless
X             LTM Eastings          (float,int) meters
Y             LTM Northings         (float,int) meters

Raises:
ValueError: If coordinates are in polar regions or not in 25km range"""

# convert to float is value was zero padded
if isinstance(E, str):
   E = float(E)
   N = float(N)

if process_errors:
   if E > 25000:
      raise ValueError("Operation aborted: Easting greater than 25km.")
   elif E < 0:
      raise ValueError("Operation aborted: Easting grid coordinate negative.")
   elif N > 25000:
      raise ValueError("Operation aborted: Northing greater than 25km")
   elif N < 0:
      raise ValueError("Operation aborted: Northing grid coordinate negative")
   elif lonBand < 1:
      raise ValueError("Operation aborted: Incorrect LTM zone formatting")
   elif lonBand > 45:
      raise ValueError("Operation aborted: Incorrect LTM zone formatting")
   elif latBand in ["A","B","Y","Z"]:
      raise ValueError("Operation aborted: LGRS coordinate is polar")

# letter set for latitude zones and 25km grid zones
# determined above.

# number of unique letter sets for LGRS
cols = 1
rows = 3

# recover the hemisphere
h = 'N' if latBand > 'N' else 'S' # ternary argument assignment to
                        # determine the hemisphere

# recover easting specified by e25k | +5 To center the letter reference
col = e25kLetters[int((lonBand-1)%cols)].index(e25k) + 5
```

```
    e25kNum = col * 25E3 # add back 25k to easting in meters

    # recover northing specified by n25k
    row = n25kLetters[int((lonBand-1)%rows)].index(n25k)
    n25kNum = row * 25E3 # add back n25k in meters

    # get bottom of latitude band
    latBand = (latBands.index(latBand)-11)*8 # converts to degrees

    # get northing off of bottom of band
    nBand=np.floor(toLTM(ZoneWidth,latBand,zone=None,trunc_val=0)[3]/25E3)*25E3

    # 25km grid square row letters repeat every 500km heading north.
    # Iteratively add back grid letter blocks to scale data to correct position
    n2M = 0
    while (n2M + n25kNum + N < nBand):
        n2M += 500E3

    # reconstruct positional values
    E=e25kNum + E
    N=n2M + n25kNum + N

    return lonBand, h, E, N

# ---------------------------------------------------------------------
# SECTION 2.23:  Convert to LPS from LGRS


def LGRStoLPS(lonBand, e25k, n25k, E, N, process_errors=True):
    """Converts LGRS coordinate of longitude band (LTM zone),latitude
    band, 25km easting grid letter, 25km northing grid letter, Easting
    in 25km grid zone, and Northing in 25km grid zone to LTM Easting
    and Northing Coordinates. All LGRS coordinates are required for proper
    output.
    If coordinate is truncated previously there will be some precision
    loss when converting back to LatLon.

    Parameters:

    lonBand        LGRS polar zone (A,B,Y,Z)     (str)
    e25k           25km grid zone easting letter (str)
    n25k           25km grid zone northing letter (str)
    E              25km grid zone easting (str,float) meters
    N              25km grid zone northing (str,float) meters

    Returns:
    h              hemisphere             (str) unitless
    X              LPS Eastings           (float,int) meters
    Y              LPS Northings          (float,int) meters

    Raises:
    ValueError: If coordinates are in polar regions or not in 25km range"""

    # convert to float if value was zero padded
    if isinstance(E, str):
        E = float(E)
```

```
        N = float(N)

    if process_errors:
        if E > 25000:
            raise ValueError("Operation aborted: Easting greater than 25km.")
        elif E < 0:
            raise ValueError("Operation aborted: Easting grid coordinate negative.")
        elif N > 25000:
            raise ValueError("Operation aborted: Northing greater than 25km")
        elif N < 0:
            raise ValueError("Operation aborted: Northing grid coordinate negative")
        elif lonBand not in ["A","B","Y","Z"]:
            raise ValueError("Operation aborted: LGRS coordinate is not polar")

    # recover hemisphere
    if lonBand == 'A' or lonBand == 'B':
        h = 'S'
    elif lonBand == 'Y' or lonBand =='Z':
        h = 'N'
    else:
        raise ValueError("Operation aborted: LGRS hemisphere not correct")

    # recover location of 25km grid square lower left corner
    # shifted increases from A to the East. Decrements from Z to west
    if lonBand=='A' or lonBand=='Y':

        # this is the member of cells between center
        # and the cell in question
        col=(len(e25kLetters_polar)-1)-e25kLetters_polar.index(e25k)+1
        e25kNum = -1*col * 25E3 # add back 25km to easting in meters
        # E=25E3-E
    else:
        col = e25kLetters_polar.index(e25k)
        e25kNum = col * 25E3 # add back 25km to easting in meters

    # recover northing specified by n25k

    # added for extended range characters
    if n25k=='-':
        row=-1-(len(n25kLetters_polar)-1)//2-1
    elif n25k=='+':
        row=len(n25kLetters_polar)-(len(n25kLetters_polar)-1)//2-1
    else:
        # n25k = n25kLetters_polar[row]
        row = n25kLetters_polar.index(n25k)-(len(n25kLetters_polar)-1)//2-1

    n25kNum = row * 25E3 #add back n25k in meters

    # reconstruct relative positional values and add back false origin
    E = e25kNum + E + FalseEasting_polar
    N = n25kNum + N + FalseNorthing_polar

    return h, E, N

# ----------------------------------------------------------------------
# SECTION 2.24: Convert to LGRS in ACC format from LTM
```

```
def toLGRS_ACC(X, Y, lonBand, h, trunc_val=10, ACC=True, process_errors=True):
    “””Converts LTM Easting and Northing Coordinates to LGRS
    in Artemis Condensed Coordinate (ACC) format. Coordinates are
    a longitude band (LTM zone),latitude band, 25km easting grid letter,
    25k northing grid letter, 1km easting grid letter, 1 km northing
    grid letter, Easting in 1km grid zone truncated to 10m, and
    Northing in 1km grid zone truncated to 10m.
    All LTM coordinates are required for proper output.
    As the coordinate is truncated to 10m the coordinate will
    express some round off during the inverse conversions.
    ACC does not have to be truncated, however, the coordinate
    length will be larger than 6 characters.

    Parameters:
    X              LTM Eastings           (float,int) meters
    Y              LTM Northings          (float,int) meters
    lonBand        LTM zone               (int,float) scaler
    h              hemisphere             (str) unitless
    trunc_val      precision              (int,float): meters
    process_errors  Processing flag       (logical): boolean

    Returns:
    lonBand        LGRS zone (LTM zone)    (float) scaler
    latBand        C-X 8° latitude reference (str)
    e25k           25km grid zone easting letter (str)
    n25k           25km grid zone northing letter (str)
    e1k            1km grid zone easting letter (str)
    n1k            1km grid zone easting letter (str)
    E              1km grid zone easting (str) meters
    N              1km grid zone northing (str) meters

    Raises:
    ValueError: If coordinates are not within a the proper LTM range of,
            125,000m ≤ E < 375,000m
            0m ≤ N < 2,500,000m
            or has the correct zone or if coordinate ACC truncation
            value is not correct
            “””

    if process_errors:
        if X > 375000:
            raise ValueError(“Operation aborted: Easting less than 375km.”)
        elif X < 125000:
            raise ValueError(“Operation aborted: Easting less than 125km.”)
        elif Y > 2500000:
            raise ValueError(“Operation aborted: Northing exceeds 2,500km”)
        elif Y < 0:
            raise ValueError(“Operation aborted: Northing negative”)
        elif lonBand < 1:
            raise ValueError(“Operation aborted: Incorrect LTM zone formatting”)
        elif lonBand > 45:
            raise ValueError(“Operation aborted: Incorrect LTM zone formatting”)
        elif ACC == True and trunc_val != 10:
            raise ValueError(“Operation aborted: ACC truncation precision “
```

```
                "incorrect. Change to 10m.")

# letter set for latitude zones, 25km, and 1km grid zones
# determined above.

# number of unique letter sets for LGRS
cols = 1
rows = 3

# global values | false easting and northing values must be applied

# convert UTM to LatLon to get latitude to determine the correct band
_,lat = toLatLon(X,Y,lonBand,h,'Lunar')

# grid zones are 8° tall, at 0° N is 10th band: only works w/ 8°
latBand = latBands[int(np.floor(lat/(2*ZoneWidth)+len(latBands)/2))]

# 25km zones
col = int(np.floor(X/25E3)) # determine Easting letter index
row = int(np.floor(Y/25E3) % 20) # determine Northing letter index

# assign Easting letter by index. Five is subtracted to center the
# letter reference
e25k = e25kLetters[int((lonBand-1)%cols)][col-5]

# assign Northing letter by index
# rows in zones are A-V, F-E, or L-K.
n25k = n25kLetters[int((lonBand-1)%rows)][row]

# truncate easting/northing to within 25km grid square
E = X % 25E3
N = Y % 25E3

# generate 1k lettering
ACC_col = int(np.floor(E/1E3)%25)
e1k = e1kmLetters[ACC_col]

# generate 1k lettering
ACC_row = int(np.floor(N/1E3)%25)
n1k = n1kmLetters[ACC_row]

# truncate final easting/northing to within 1km grid square
E %= 1E3
N %= 1E3

# add zero padding to value. Convert to strings to retain vals
if trunc_val != None:

    if ACC and trunc_val==10: # apply ACC formatting

        # truncate value to 10m
        E = trunc(E, trunc_val, process_errors=True)
        N = trunc(N, trunc_val, process_errors=True)

        # convert to string and apply zero padding
        E = "{:05.0f}".format(E)
```

```
            N = "{:05.0f}".format(N)

            # remove last value
            E = E[2:-1]
            N = N[2:-1]

            return e1k, E, n1k, N

        else:
            # truncate to format final output
            E = trunc(E, trunc_val, process_errors=True)
            N = trunc(N, trunc_val, process_errors=True)

            # add zero padding
            if trunc_val!=0:
                E = "{:03.0f}".format(E)
                N = "{:03.0f}".format(N)


        return lonBand, latBand, e25k, n25k, e1k, E, n1k, N

# ----------------------------------------------------------------------
# SECTION 2.25: Convert to LGRS in ACC format from LPS


def toLGRS_polar_ACC(E, N, h, trunc_val=10, ACC=True, process_errors=True):
    """Converts LTM Easting and Northing Coordinates to LGRS coordinate
    in ACC format. Coordinates are a longitude band, 25k easting
    grid letter, 25k northing grid letter, 1km easting grid letter,
    1km northing grid letter, Easting in 1km grid zone truncated to 10m,
    and Northing in 1km grid zone truncated to 10m.

    If coordinate is truncated then the coordinate is zero padded
    to the proper string length. If not truncated, the final coordinate
    is left as a floating point.

    Parameters:
    E             LPS Eastings          (float,int) meters
    N             LPS Northings         (float,int) meters
    h             hemisphere            (str) unitless
    trunc_val     precision             (int,float): meters
    process_errors  Processing flag     (logical): boolean

    Returns:
    lonBand       LGRS zone A,B,Y,Z    (str) scaler
    e25k          25km grid zone easting letter (str)
    n25k          25km grid zone northing letter (str)
    e1k           1km grid zone easting letter (str)
    n1k           1km grid zone northing letter (str)
    E             1km grid zone easting (str,float) meters
    N             1km grid zone northing (str,float) meters

    Raises:
    ValueError: If coordinates are not within a the proper LPS range of,
            197,000m ≤ E < 805000m
            197,000m ≤ N < 805000m
```

```
        Program terminates if coordinates are in the wrong zone
        or if the precision for ACC format is not correct.
        “””
if process_errors:
    if E > 805000:
        raise ValueError(“Operation aborted: Easting outside of value range.”)
    elif E < 197000:
        raise ValueError(“Operation aborted: Easting outside of value range.”)
    elif N > 805000:
        raise ValueError(“Operation aborted: Northing outside of value range.”)
    elif N < 197000:
        raise ValueError(“Operation aborted: Northing outside of value range”)
    elif h not in [“S”, “N”]:
        raise ValueError(“Operation aborted: Wrong hemisphere provided”)
    elif ACC == True and trunc_val != 10:
        raise ValueError(“Operation aborted: ACC truncation precision “
                “incorrect. Change to 10m.”)
# lat_band
# there is no latitude band on the poles

# lon_Band
# determine the longitudinal LGRS zone with logical checks
if h == ‘S’:
    if E < FalseEasting_polar:
        lonBand = ‘A’
    elif E >= FalseEasting_polar:
        lonBand = ‘B’
elif h == ‘N’:
    if E < FalseEasting_polar:
        lonBand = ‘Y’
    elif E >= FalseEasting_polar:
        lonBand = ‘Z’

# letter set for 25km and 1km grid zones determined above.

# make coordinates relative to false northing or easting
X = E - FalseEasting_polar
Y = N - FalseNorthing_polar

# determine number for letter index center
centX = (len(e25kLetters_polar) - 1)//2 # this is index 13
centY = (len(n25kLetters_polar) - 1)//2 #

# shifted increases from A to the East. Decrements from Z to west
if lonBand == ‘A’ or lonBand == ‘Y’:
    col = int((len(n25kLetters_polar) - 1) - (np.floor(np.abs(X)/25E3)))
else:
    col = int((np.floor(X/25E3)))
e25k = e25kLetters_polar[col]

# add half the letter width to the index to correct position
# done b/c everything is relative to the center
row = int(np.floor(Y/25E3) + centY) + 1

# added extended range characters
if row < 0:
```

```
    n25k = '-'
elif row > len(n25kLetters_polar) - 1:
    n25k = '+'
else:
    n25k = n25kLetters_polar[row]

# truncate easting/northing to within 25km grid square
# values are reassigned here
E = np.abs(X) % 25E3
N = Y % 25E3

# Adjustment to the easting at the 1km level
if lonBand == 'A' or lonBand == 'Y':
    LGRS_col = int((np.floor((25E3 - np.abs(E))/1E3))%25)
else:
    LGRS_col = int(np.floor((E/1E3)%25))

e1k = e1kmLetters[LGRS_col]

# generate 1km northing lettering
LGRS_row = int((((np.floor(N/1E3)))))
n1k = n1kmLetters[LGRS_row]

if lonBand == 'A' or lonBand == 'Y':
    E = (25E3 - np.abs(E))%1E3
else:
    E %= 1E3
N %= 1E3

if trunc_val!=None:
    if ACC and trunc_val == 10: # apply ACC formatting

        # truncate value to 10m
        E = trunc(E, trunc_val, process_errors=True)
        N = trunc(N, trunc_val, process_errors=True)

        # convert to string and apply zero padding
        E = "{:05.0f}".format(E)
        N = "{:05.0f}".format(N)

        # remove last value
        E = E[2:-1]
        N = N[2:-1]

        return e1k, E, n1k, N

    else:
        # truncate to format final output
        E = trunc(E, trunc_val, process_errors=True)
        N = trunc(N, trunc_val, process_errors=True)

        # add zero padding
        if trunc_val != 0:
            E = "{:03.0f}".format(E)
            N = "{:03.0f}".format(N)
    # print(lonBand, e25k, n25k, e1k, E, n1k, N)
```

```
    return lonBand, e25k, n25k, e1k, E, n1k, N

# ----------------------------------------------------------------------
# SECTION 2.26: Convert to LTM form LGRS in ACC format


def LGRS_ACCtoLTM(latBand, lonBand, e25k, n25k, e1k, E, n1k, N,
                                ACC=True,process_errors=True):
  """Converts LGRS in ACC format of longitude band (LTM zone),latitude
  band, 25k easting grid letter, 25k northing grid letter,
  1k easting grid letter, 1k northing grid letter, Easting in 1km grid zone,
  and Northing in 1km grid zone to LTM Easting and Northing Coordinates.

  All LGRS coordinates are required for proper output.
  If coordinate is truncated previously there will be some precision
  loss when converting back to LatLon.
  Conversion requires 2LTM function.

  Parameters:
  lonBand        LGRS zone (LTM zone)    (float) scaler
  latBand        C-X 8° latitude reference (str)
  e25k           25km grid zone easting letter (str)
  n25k           25km grid zone northing letter (str)
  e1k            1km grid zone easting letter (str)
  n1k            1km grid zone easting letter (str)
  E              1km grid easting (str) meters
  N              1km grid northing (str) meters

  Returns:
  X              LTM Eastings           (float,int) meters
  N              LTM Northings          (float,int) meters
  lonBand        LTM zone               (int,float) scaler
  h              hemisphere             (str) unitless
  trunc_val      precision              (int,float): meters
  process_errors  Processing flag        (logical): boolean

  Raises:
  ValueError: If coordinates are in polar regions or not in 25km range,
          or grid area."""

  # convert to float is value was zero padded or in ACC format
  if ACC:
    E = float(E + "0") # add back ones position and convert to float
    N = float(N + "0") # for processing

  elif isinstance(E, str):
    E = float(E)
    N = float(N)

  if process_errors:
    if E > 25000:
      raise ValueError("Operation aborted: Easting greater than 25km.")
    elif E < 0:
      raise ValueError("Operation aborted: Easting grid coordinate negative.")
    elif N > 25000:
      raise ValueError("Operation aborted: Northing greater than 25km")
```

```
        elif N < 0:
            raise ValueError("Operation aborted: Northing grid coordinate negative")
        elif lonBand < 1:
            raise ValueError("Operation aborted: Incorrect LTM zone formatting")
        elif lonBand > 45:
            raise ValueError("Operation aborted: Incorrect LTM zone formatting")
        elif latBand in ["A","B","Y","Z"]:
            raise ValueError("Operation aborted: LGRS coordinate is polar")

    # letter set for latitude zones, 25km, and 1km grid zones
    # determined above.

    # number of unique letter sets for LGRS
    cols = 1
    rows = 3

    # recover the hemisphere
    h = 'N' if latBand > 'N' else 'S' # ternary argument assignment to
                            # determine the hemisphere
    # recover 1km grid spacing
    AGRS_col = e1kmLetters.index(e1k)
    e1kNum = AGRS_col*1E3

    AGRS_row = n1kmLetters.index(n1k)
    n1kNum = AGRS_row*1E3

    # recover easting specified by e25k | +5 To center the letter reference
    col = e25kLetters[int((lonBand-1)%cols)].index(e25k) + 5
    e25kNum = col * 25E3 # add back 25k to easting in meters

    # recover northing specified by n25k
    row = n25kLetters[int((lonBand - 1)%rows)].index(n25k)
    n25kNum = row * 25E3 # add back n25k in meters

    # get bottom of latitude band
    latBand = (latBands.index(latBand) - 11)*8 # converts to degrees

    # get northing off of bottom of band,
    nBand=np.floor(toLTM(ZoneWidth,latBand,zone=None,trunc_val=0,process_errors=False)[3]/25E3)*25E3

    # 25km grid square, row letters repeat every 500km heading north.
    # iteratively add back grid letter blocks to scale data to correct position
    n2M = 0
    while (n2M + n25kNum + N < nBand):
        n2M += 500E3

    # reconstruct positional values
    E = e25kNum + e1kNum + E
    N = n2M + n25kNum + n1kNum + N

    return lonBand, h, E, N

# ----------------------------------------------------------------------
# SECTION 2.27: Convert to LPS from LGRS in ACC format
```

```
def LGRS_ACCtoLPS(lonBand, e25k, n25k, e1k, E, n1k, N,
                                ACC=True, process_errors=True):
    """Converts LGRS ACC formatted grid coordinates to LPS coordinates.

    If coordinate is truncated previously there will be some precision
    loss when converting back to LatLon.

    Parameters:
    lonBand        LGRS zone A,B,Y,Z    (str) scaler
    e25k           25km grid zone easting letter (str)
    n25k           25km grid zone northing letter (str)
    e1k            1km grid zone easting letter (str)
    n1k            1km grid zone northing letter (str)
    E              1km grid zone easting (str,float) meters
    N              1km grid zone northing (str,float) meters

    Returns:
    E              LPS Eastings           (float,int) meters
    N              LPS Northings          (float,int) meters
    h              hemisphere             (str) unitless
    trunc_val      precision              (int,float): meters
    process_errors  Processing flag       (logical): boolean

    Raises:
    ValueError: If coordinates are in polar regions or not in 25km range"""

    # convert to float is value was zero padded
    if ACC:
        E = float(E + "0") # add back ones position and convert to float
        N = float(N + "0") # for processing

    elif isinstance(E, str):
        E = float(E)
        N = float(N)

    if process_errors:
        if E > 25000:
            raise ValueError("Operation aborted: Easting greater than 25km.")
        elif E < 0:
            raise ValueError("Operation aborted: Easting grid coordinate negative.")
        elif N > 25000:
            raise ValueError("Operation aborted: Northing greater than 25km")
        elif N < 0:
            raise ValueError("Operation aborted: Northing grid coordinate negative")
        elif lonBand not in ["A","B","Y","Z"]:
            raise ValueError("Operation aborted: LGRS coordinate is not polar")

    # recover hemisphere
    if lonBand == 'A'or lonBand == 'B':
        h = 'S'
    elif lonBand == 'Y'or lonBand == 'Z':
        h = 'N'
    else:
        raise ValueError("Operation aborted: LGRS hemisphere not correct")

    # recover location of 25km grid square lower left corner
```

```
    # shifted increases from A to the East. Decrements from Z to west

    centX = (len(e25kLetters_polar) - 1)//2
    centY = (len(n25kLetters_polar) - 1)//2

    # recover location of 25km grid square lower left corner
    # shifted increases from A to the East. Decrements from Z to west
    if lonBand==’A’ or lonBand==’Y’:

        # this is the member of cells between center
        # and the cell in question
        col = (len(e25kLetters_polar) - 1) - e25kLetters_polar.index(e25k) + 1
        e25kNum = -1*col * 25E3 # add back 25km to easting in meters
        # E=25E3-E
    else:
        col = e25kLetters_polar.index(e25k)
        e25kNum = col * 25E3 # add back 25km to easting in meters

    # added for extended range characters
    if n25k == ‘-’:
        row = -1 - (len(n25kLetters_polar) -1 )//2 -1
    elif n25k==’+’:
        row=len(n25kLetters_polar) - (len(n25kLetters_polar) - 1)//2 -1
    else:
        row = n25kLetters_polar.index(n25k) - (len(n25kLetters_polar) - 1)//2 -1

    n25kNum = row * 25E3 #add back n25k in meters

    # determine 1km spacing from letters
    LGRS_col = e1kmLetters.index(e1k)
    e1kNum = LGRS_col*1E3

    LGRS_row = n1kmLetters.index(n1k)
    n1kNum = LGRS_row*1E3

    # reconstruct relative positional values and  add back false origins
    E = e25kNum + e1kNum + E + FalseEasting_polar
    N = n25kNum + n1kNum + N + FalseNorthing_polar

    return h, E, N

# ---------------------------------------------------------------------
# SECTION 2.28: Convert planetocentric latitude to colatitude


def planetocentric_lat_degrees(colat):
    “””convert points from colatitude 0-180 to 90--90 degrees.
    Parameters
    colat   0-180 longitude (float): degrees

    Returns
    lat  -90-90 latitude (float): degrees

    “””
    return 90. - colat
```

```
# ----------------------------------------------------------------------
# SECTION 2.29: Convert planetocentric colatitude to latitude
def planetocentric_colat_degrees(lat):
    """convert points from colatitude 90--90 to 0-180 degrees.
    Parameters
    colat 0-180 longitude (float): degrees

    Returns
    lat  -90-90 latitude (float): degrees
    """
    return -1*(lat - 90.)

# ----------------------------------------------------------------------
# SECTION 2.30: Convert planetocentric longitude to colongitude


def planetocentric_colon_degrees(lon): # 0-360
    """convert points from -180 to 180 to 0-360 degrees.
    Parameters
    lon   -180-180 longitude (float): degrees

    Returns
    colon   0-360 longitude (float): degrees
    """
    return (lon) % 360.

# ----------------------------------------------------------------------
# SECTION 2.31: Convert planetocentric colongitude to longitude


def planetocentric_lon_degrees(colon): # -180 - 180
    """convert points from 0-360 degrees to -180 to 180.
    Parameters
    colon   0-360 longitude (float): degrees
    """
    return ((colon + 540.)%360) - 180.

# ----------------------------------------------------------------------
# SECTION 2.32: Convert degrees to decimal minutes


def degrees_to_decimal_minutes(degrees):
    """converts degrees to decimal minutes:
    Parameters:
    degrees       (float,int): degrees

    Returns:
    degrees       (float,int): decimal degrees
    minutes       (float,int): degrees(min)"""

    # Extract the integer part (deg) and the fractional part (min)
    deg = int(degrees)
    minutes = (degrees - deg) * 60

    return deg, minutes
```

```
# ----------------------------------------------------------------------
# SECTION 2.33: Convert decimal minutes to degrees


def decimal_minutes_to_degrees(degrees, minutes):
    """converts decimal minutes to degrees:
    Parameters:
    degrees      (float,int): decimal degrees
    minutes      (float,int): degrees(min)

    Returns:
    degrees      (float,int): degrees"""

    # Calculate the degrees in decimal format
    degrees_decimal = degrees + minutes / 60

    return degrees_decimal

# ----------------------------------------------------------------------
# SECTION 2.34: Convert degrees to decimal seconds


def degrees_to_decimal_seconds(degrees):
    """converts  degrees to decimal seconds :
    Parameters:
    degrees (float): decimal degrees

    Returns:
    degrees      (float,int): degrees
    minutes      (float,int): degrees(min)
    seconds      (float): degrees(sec)"""

    # Extract the integer part (deg) and the fractional part (min)
    deg = int(degrees)
    minutes_decimal = (degrees - deg) * 60

    # Extract the integer part (min) and the fractional part (sec)
    minutes = int(minutes_decimal)
    seconds = (minutes_decimal - minutes) * 60

    return deg, minutes, seconds

# ----------------------------------------------------------------------
# SECTION 2.35: Convert decimal seconds to degrees


def decimal_seconds_to_degrees(degrees, minutes, seconds):
    """converts decimal seconds to degrees:
    degrees      (float,int): degrees
    minutes      (float,int): degrees(min)
    seconds      (float,int): degrees(sec)

    Returns:
    float: decimal degrees"""

    # Calculate the degrees in decimal format
```

```
    degrees_decimal = degrees + minutes / 60 + seconds / 3600

    return degrees_decimal

# ----------------------------------------------------------------------
# SECTION 2.36: Formula to truncate coordinates
# If difference between coordinate and nearest whole number is less than
# 1mm, ie .001 or .999 we round


def check_decimal_round(value, tolerance=0.001):
    """Assists the truncation function on the south pole.
    often the truncation value forces a round down because of the
    roundoff error. This dependence forces a round if the coordinate
    is within 1mm of the nearest whole value coordinate.
    Parameters:
    val       (float,int): meters
    tolerance (float,int): meters

    Returns:
    val, (float): The result of rounding if within 1mm
    """
    rounded_value = round(value)
    if abs(value - rounded_value) < tolerance:
        return round(value)
    else:
        return value

def trunc(val, precision,process_errors=True):
    """Truncates value to the desired precision
    Parameters:
    val       (float,int): meters
    precision (float,int): meters

    Returns:
    float,int: The result of the truncation operation.
            Value not processed if precision is 0

    Raises:
    ValueError: If the precision is not a multiple of 10."""

    if process_errors:
        if precision not in [0,1,10,100,1000,10000,
                             100000,1000000,10000000]:
            raise ValueError("Operation Aborted: Truncation is only "
                             "allowed for multiples of 10.")

    if precision == 0:
        return val
    else:
        val=check_decimal_round(val, tolerance=0.001)
        return float(int(val/precision)*precision)


# ======================================================================
# SECTION 3: MAIN PROGRAM
```

```
    # SECTION 3.1: Importing system variables for processing
    # SECTION 3.2: Converting Coordinates
    # SECTION 3.2.1: LATLON CONVERSIONS
        # SECTION 3.2.2: LTM CONVERSIONS
        # SECTION 3.2.3: LPS CONVERSIONS
        # SECTION 3.2.4: LGRS CONVERSIONS
        # SECTION 3.2.4: LGRS ACC CONVERSIONS
    # SECTION 3.3: Exporting data

# ----------------------------------------------------------------------
# SECTION 3.1: Importing system variables for processing


def main(switch, trunc_val, condensed=True):
    “”” Complete conversions between LTM, LPS, LGRS, LGRS in ACC, and
        LatLon. This function is designed to accommodate shell scripting,
        All other equations should be able to be utilized in Python.
        Specify switch statement variable to continue:\n”LatLon2LTM\n”
        “LatLon2LPS\n””LatLon2LGRS\n””LatLon2PolarLGRS\n”
        “LatLon2LGRS_ACC\n””LatLon2PolarLGRS_ACC\n””LatLon2ACC\n”
        “LatLon2Polar_ACC\n””LTM2LatLon\n””LTM2LGRS\n””LTM2LGRS_ACC\n”
        “LTM2ACC\n””LPS2LatLon\n””LPS2PolarLGRS\n””LPS2PolarLGRS_ACC\n”
        “LPS2ACC\n””LGRS2LTM\n””PolarLGRS2LPS\n””LGRS2LatLon\n”
        “PolarLGRS2LatLon\n””LGRS_ACC2LatLon\n””PolarLGRS_ACC2LatLon\n”
        “LGRS_ACC2LTM\n””PolarLGRS_ACC2LPS\n”)”””

    #  argument formats
    form_in = switch.split(“2”)[0] #input argument format
    form_out = switch.split(“2”)[1] # output argument format

    # input coordinates
    try:
        if condensed:
            if  form_in == “LatLon”:
                if len(sys.argv)-1 == 3:
                    lat = float(sys.argv[2])
                    lon = float(sys.argv[3])
                else:
                    raise IndexError(“Operation aborted: Number of inputs not correct.”
                                “Input for LatLon:\nLat\nLon”)

            # LTM:       zone,h,E,N,
            elif form_in == “LTM”:
                if len(sys.argv)-1 == 2:
                    zone, h, E, _, N, _ = list(re.findall(LTM_pattern,sys.argv[2])[0])
                    zone = int(zone)
                    h = str(h)
                    E = float(E)
                    N = float(N)

                else:
                    raise IndexError(“Operation aborted: Number of inputs not correct.”
                                “Input for Condensed LTM: “
                                “\nzone\{1-45\}\nh\{N|S\}\n00000E\n00000N.”)
            # LPS:       h,E,N
            elif form_in == “LPS”:
```

```
    if len(sys.argv)-1 == 2:
        h, E, _, N, _ = list(re.findall(LPS_pattern,sys.argv[2])[0])
        h = str(h)
        E = float(E)
        N = float(N)

    else:
        raise IndexError("Operation aborted: Number of inputs not correct."
                         "Input for Condensed LPS:"
                         "\nh\{N|S\}\n00000E\n00000N.")

# LGRS:       lonBand,latBand,e25k,n25k,E,N
elif form_in=="LGRS":
    if len(sys.argv)-1 == 2:
        lonBand, latBand, e25k, n25k, E, N = list(re.findall(LGRS_pattern,sys.argv[2])[0])
        lonBand = int(lonBand)
        latBand = str(latBand)
        e25k = str(e25k)
        n25k = str(n25k)
        E = float(E)
        N = float(N)
    else:
        raise IndexError("Operation aborted: Number of inputs not correct."
                         "Input for Condensed LGRS: "
                         "\nlonBand\{1-45\}\nlatBand\{N|S\}\n"
                         "e25k\{A-Z\}\nn25k\{A-Z\}\nE{00000}\nN{00000}")
# Polar_LGRS  lonBand,e25k,n25k,E,N
elif form_in == "PolarLGRS":
    if len(sys.argv)-1 == 2:
        lonBand, e25k, n25k, E, N = list(re.findall(PolarLGRS_pattern,sys.argv[2])[0])

        lonBand = str(lonBand)
        e25k = str(e25k)
        n25k = str(n25k)
        E = float(E)
        N = float(N)

    else:
        raise IndexError("Operation aborted: Number of inputs not correct."
                         "Input for Condensed Polar LGRS: "
                         "\nlatBand\{N|S\}\ne25k\{A-Z\}\nn25k\{A-Z\}"
                         "\nE{00000}\nN{00000}")
# LGRS_ACC:         lonBand,latBand,e25k,n25k,e1k,E,n1k,N
elif form_in == "LGRS_ACC":
    if len(sys.argv)-1 == 2:
        lonBand, latBand, e25k, n25k, e1k, E, n1k, N=list(re.findall(LGRS_ACC_pattern,sys.argv[2])[0])

        lonBand = int(lonBand)
        latBand = str(latBand)
        e25k = str(e25k)
        n25k = str(n25k)
        e1k = str(e1k)
        E = float(E)
        n1k = str(n1k)
        N = float(N)
    else:
```

```
            raise IndexError("Operation aborted: Number of inputs not correct."
                             "Input for Condensed LGRS_ACC: "
                        "\nlonBand\{1-45\}\nlatBand\{N|S\}\n"
                        "e25k\{A-Z\}\nn25k\{A-Z\}\nE{00000}\nN{00000}"
            "Input for LGRS_ACC:\nlonBand\nlatBand\ne25k\nn25k\ne1k\nn1k\nE\nN")

        elif form_in == "PolarLGRS_ACC":
            if len(sys.argv)-1 == 2:
                lonBand, e25k, n25k, e1k, E, n1k, N=list(re.findall(PolarLGRS_ACC_pattern,sys.argv[2])[0])

                lonBand=str(lonBand)
                e25k=str(e25k)
                n25k=str(n25k)
                e1k=str(e1k)
                E=float(E)
                n1k=str(n1k)
                N=float(N)
            else:
                raise IndexError("Operation aborted: Number of inputs not correct."
                "Input for PolarLGRS_ACC:\nlonBand\ne25k\nn25k\ne1k\nn1k\nE\nN")
        else:
            raise IndexError("Operation aborted: Input failed")

    elif not condensed:
        # LatLon:     lon,lat,
        if  form_in == "LatLon":
            if len(sys.argv)-1 == 3:
                lat=float(sys.argv[2])
                lon=float(sys.argv[3])
            else:
                raise IndexError("Operation aborted: Number of inputs not correct."
                             "Input for LatLon:\nLat\nLon")
        # LTM:        zone,h,E,N,
        elif form_in == "LTM":
            if len(sys.argv)-1 == 5:
                zone = int(sys.argv[2])
                h = str(sys.argv[3])
                E = float(sys.argv[4])
                N = float(sys.argv[5])
            else:
                raise IndexError("Operation aborted: Number of inputs not correct."
                             "Input for LTM:\nzone\nh\nE\nN")
        # LPS:        h,E,N
        elif form_in == "LPS":
            if len(sys.argv)-1 == 4:
                h = str(sys.argv[2])
                E = float(sys.argv[3])
                N = float(sys.argv[4])
            else:
                raise IndexError("Operation aborted: Number of inputs not correct."
                             "Input for LPS:\nh\nE\nN")
        # LGRS:       lonBand,latBand,e25k,n25k,E,N
        elif form_in == "LGRS":
            if len(sys.argv)-1 == 7:
                lonBand = int(sys.argv[2])
                latBand = str(sys.argv[3])
```

```
            e25k = str(sys.argv[4])
            n25k = str(sys.argv[5])
            E = float(sys.argv[6])
            N = float(sys.argv[7])
        else:
            raise IndexError("Operation aborted: Number of inputs not correct."
                    "Input for LGRS:\nlonBand\nlatBand\ne25k\nn25k\nE\nN")
    # Polar_LGRS  lonBand,e25k,n25k,E,N
    elif form_in == "PolarLGRS":
        if len(sys.argv)-1 == 6:
            lonBand = str(sys.argv[2])
            e25k = str(sys.argv[3])
            n25k = str(sys.argv[4])
            E = float(sys.argv[5])
            N = float(sys.argv[6])
        else:
            raise IndexError("Operation aborted: Number of inputs not correct."
                    "Input for PolarLGRS:\nlonBand\ne25k\nn25k\nE\nN")
    # LGRS_ACC:         lonBand,latBand,e25k,n25k,e1k,E,n1k,N
    elif form_in == "LGRS_ACC":
        if len(sys.argv)-1 == 9:
            lonBand = int(sys.argv[2])
            latBand = str(sys.argv[3])
            e25k = str(sys.argv[4])
            n25k = str(sys.argv[5])
            e1k = str(sys.argv[6])
            E = float(sys.argv[7])
            n1k = str(sys.argv[8])
            N = float(sys.argv[9])
        else:
            raise IndexError("Operation aborted: Number of inputs not correct."
            "Input for LGRS_ACC:\nlonBand\nlatBand\ne25k\nn25k\ne1k\nn1k\nE\nN")
    elif form_in == "PolarLGRS_ACC":
        if len(sys.argv)-1 == 8:
            lonBand = str(sys.argv[2])
            e25k = str(sys.argv[3])
            n25k = str(sys.argv[4])
            e1k = str(sys.argv[5])
            E = float(sys.argv[6])
            n1k = str(sys.argv[7])
            N = float(sys.argv[8])

        else:
            raise IndexError("Operation aborted: Number of inputs not correct."
            "Input for PolarLGRS_ACC:\nlonBand\ne25k\nn25k\ne1k\nn1k\nE\nN")
    else:
        raise IndexError("Operation aborted: Input failed")
else:
    raise IndexError("Coordinate style could not be determined."
                "Please specify if coordinates are condensed "
                "or not. (Note this is not ACC formatting).")

except IndexError as e:
    exc_type, exc_obj, exc_tb = sys.exc_info()
    fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
    print(exc_type, fname, exc_tb.tb_lineno)
```

```
        print(e)
        exit(1)



# ---------------------------------------------------------------------
# SECTION 3.2: Converting Coordinates

    try:
        # SECTION 3.2.1: LATLON CONVERSIONS
        # Convert Planetocentric LatLon to LTM
        if switch == “LatLon2LTM”:

            zone, h, E, N = toLTM(lon, lat,
                        zone=None,
                        trunc_val=trunc_val,
                        process_errors=True)

        # Convert Planetocentric LatLon to LPS
        elif switch == “LatLon2LPS”:

            h, E, N = toLPS(lon, lat,
                    trunc_val=trunc_val,
                    eqs=”Spherical”,
                    process_errors=True)

        # Convert Planetocentric LatLon to LGRS
        elif switch == “LatLon2LGRS”:

            zone, h, E1, N1 = toLTM(lon, lat,
                        zone=None,
                        trunc_val=0,
                        process_errors=True)

            lonBand, latBand, e25k, n25k, E, N=toLGRS(E1, N1, zone, h,
                                    trunc_val=trunc_val,
                                    process_errors=True)

        # Convert Planetocentric LatLon to Polar LGRS
        elif switch == “LatLon2PolarLGRS”:

            h, E1, N1 = toLPS(lon, lat,
                        trunc_val=0,
                        eqs=”Spherical”,
                        process_errors=True)

            lonBand, e25k, n25k, E, N=toLGRS_polar(E1, N1, h,
                                    trunc_val=trunc_val,
                                    process_errors=True)

        # Convert Planetocentric LatLon to LGRS in ACC
        elif switch == “LatLon2LGRS_ACC”:

            zone, h, E1, N1 = toLTM(lon, lat,
                        zone=None,
                        trunc_val=0,
```

```
                process_errors=True)

    lonBand, latBand, e25k, n25k, e1k, E, n1k, N = toLGRS_ACC(E1, N1, zone, h,
                                        trunc_val=trunc_val,
                                        ACC=False,
                                        process_errors=True)

# Convert Planetocentric LatLon to Polar LGRS in ACC
elif switch == "LatLon2PolarLGRS_ACC":

    h, E1, N1 = toLPS(lon, lat,
                trunc_val=0,
                eqs="Spherical",
                process_errors=True)

    lonBand, e25k, n25k, e1k, E, n1k, N = toLGRS_polar_ACC(E1, N1, h,
                                      trunc_val=trunc_val,
                                      ACC=False,
                                      process_errors=True)

# Convert Planetocentric LatLon to ACC
elif switch == "LatLon2ACC":

    zone, h, E1, N1 = toLTM(lon, lat,
                    zone=None,
                    trunc_val=0,
                    process_errors=True)

    e1k, E, n1k, N = toLGRS_ACC(E1, N1, zone, h,
                       trunc_val=10,
                       ACC=True,
                       process_errors=True)

# Convert Planetocentric LatLon to Polar ACC
elif switch == "LatLon2Polar_ACC":

    # ACC formatted only
    h, E1, N1 = toLPS(lon, lat,
                trunc_val=0,
                eqs="Spherical",
                process_errors=True)

    e1k, E, n1k, N = toLGRS_polar_ACC(E1, N1, h,
                trunc_val=10,
                ACC=True,
                process_errors=True)

#SECTION 3.2.2: LTM CONVERSIONS
# Convert LTM to Planetocentric LatLon
elif switch == "LTM2LatLon":

    lon, lat = toLatLon(E, N, zone, h,
                    process_errors=True)

# LTM to LGRS
elif switch == "LTM2LGRS":
```

```
    lonBand, latBand, e25k, n25k, E, N = toLGRS(E, N, zone, h,
                                    trunc_val=trunc_val,
                                    process_errors=True)

# LTM to ACC LGRS
elif switch == “LTM2LGRS_ACC”:

    lonBand, latBand, e25k, n25k, e1k, E, n1k, N = toLGRS_ACC(E, N, zone, h,
                                        trunc_val=trunc_val,
                                        ACC=False,
                                        process_errors=True)

# LTM to ACC
elif switch == “LTM2ACC”:

    e1k, E, n1k, N = toLGRS_ACC(E, N, zone, h,
                        trunc_val=10,
                        ACC=True,
                        process_errors=True)

# SECTION 3.2.3: LPS CONVERSIONS
# Convert LPS to Planetocentric LatLon
elif switch == “LPS2LatLon”:

    lon, lat = toLatLon_Polar(E, N, h,
                        eqs=”Spherical”,
                        process_errors=True)

# LPS to LGRS
elif switch == “LPS2PolarLGRS”:

    lonBand, e25k, n25k, E, N = toLGRS_polar(E, N, h,
                                trunc_val=trunc_val,
                                process_errors=True)

# LPS to LGRS ACC
elif switch == “LPS2PolarLGRS_ACC”:

    # ACC formatted
    lonBand, e25k, n25k, e1k, E, n1k, N = toLGRS_polar_ACC(E, N, h,
                                        trunc_val=trunc_val,
                                        ACC=False,
                                        process_errors=True)

# LPS to ACC
elif switch == “LPS2ACC”:

    # ACC formatted
    e1k, E, n1k, N = toLGRS_polar_ACC(E, N, h,
                            trunc_val=10,
                            ACC=True,
                            process_errors=True)

# SECTION 3.2.4: LGRS CONVERSIONS
# LGRS to Convert Planetocentric LatLon
```

```
elif switch == “LGRS2LatLon”:

   zone, h, E, N = LGRStoLTM(latBand, lonBand, e25k, n25k, E, N)

   lon, lat = toLatLon(E, N, zone, h, process_errors=True)

# LGRS to LTM
elif switch == “LGRS2LTM”:

   zone, h, E, N = LGRStoLTM(latBand, lonBand, e25k, n25k, E, N)

# LGRS to LPS
elif switch == “PolarLGRS2LPS”:

   h, E, N = LGRStoLPS(lonBand, e25k, n25k, E, N, process_errors=True)

#Convert Polar LGRS to  Planetocentric polar LatLon
elif switch == “PolarLGRS2LatLon”:

   h, E1, N1 = LGRStoLPS(lonBand, e25k, n25k, E, N, process_errors=True)

   lon, lat =toLatLon_Polar(E1, N1, h,
                     eqs=”Spherical”,
                     process_errors=True)

elif switch == “LGRS2LGRS_ACC”:

   zone, h, E1, N1 = LGRStoLTM(latBand, lonBand, e25k, n25k, E, N)

   lonBand, latBand, e25k, n25k, e1k, E, n1k, N = toLGRS_ACC(E1, N1, zone, h,
                                       trunc_val=trunc_val,
                                       ACC=False,
                                       process_errors=True)

elif switch == “LGRS2ACC”:

   zone, h, E1, N1 = LGRStoLTM(latBand, lonBand, e25k, n25k, E, N)

   e1k, E, n1k, N = toLGRS_polar_ACC(E1, N1, h,
                           trunc_val=10,
                           ACC=True,
                           process_errors=True)

elif switch == “PolarLGRS2Polar_ACC”:

   h, E1, N1 = LGRStoLPS(lonBand, e25k, n25k, E, N, process_errors=True)

   e1k, E, n1k, N = toLGRS_polar_ACC(E1, N1, h,
                           trunc_val=10,
                           ACC=True,
                           process_errors=True)

elif switch == “PolarLGRS2PolarLGRS_ACC”:

   h,E1,N1 = LGRStoLPS(lonBand, e25k, n25k, E, N, process_errors=True)
   lonBand,e25k,n25k,e1k,E,n1k,N = toLGRS_polar_ACC(E1, N1, h,
```

```
                                    trunc_val=trunc_val,
                                    ACC=False,
                                    process_errors=True)

#SECTION 3.2.4: LGRS ACC CONVERSIONS
# LGRS ACC to Convert Planetocentric LatLon
elif switch == “LGRS_ACC2LatLon”:

   zone, h, E1, N1 = LGRS_ACCtoLTM(latBand, lonBand, e25k, n25k, e1k, E, n1k, N,
                        ACC=False,
                        process_errors=True)

   lon, lat = toLatLon(E1, N1, zone, h, process_errors=True)

#  Convert Polar LGRS ACC to Planetocentric LatLon
elif switch == “PolarLGRS_ACC2LatLon”:
   h, E, N = LGRS_ACCtoLPS(lonBand, e25k, n25k, e1k, E, n1k, N,
                  ACC=False,
                  process_errors=True)
   lon, lat = toLatLon_Polar(E, N, h,
                     eqs=”Spherical”,
                     process_errors=True)

#  ACC LGRS to LTM
elif switch == “LGRS_ACC2LTM”:
   zone, h, E, N = LGRS_ACCtoLTM(latBand, lonBand, e25k, n25k, e1k, E, n1k, N,
                        ACC=False,
                        process_errors=True)

elif switch == “LGRS_ACC2LGRS”:

   zone, h, E1, N1 = LGRS_ACCtoLTM(latBand, lonBand, e25k, n25k, e1k, E, n1k, N,
                        ACC=False,
                        process_errors=True)

   lonBand, latBand, e25k, n25k, E, N = toLGRS(E1, N1, zone, h,
                              trunc_val=trunc_val,
                              process_errors=True)

# LGRS ACC to LPS
elif switch == “PolarLGRS_ACC2LPS”:

   h, E, N=LGRS_ACCtoLPS(lonBand, e25k, n25k, e1k, E, n1k ,N,
                  ACC=False,
                  process_errors=True)

elif switch == “PolarLGRS_ACC2PolarLGRS”:

   h, E1, N1 = LGRS_ACCtoLPS(lonBand, e25k, n25k, e1k, E, n1k, N,
                     ACC=False,
                     process_errors=True)

   lonBand, e25k, n25k, E, N = toLGRS_polar(E1, N1, h,
                              trunc_val=trunc_val,
                              process_errors=True)
```

```
        # flag warnings
        else:
            raise ValueError("Operation aborted: Forward Conversion Method Not Specified.")

    # print faults
    except ValueError as e:
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        print(exc_type, fname, exc_tb.tb_lineno)
        print(e)
        exit(1)


# ----------------------------------------------------------------------
# SECTION 3.3: Exporting data

    try:
        if condensed:
            if  form_out == "LatLon":
                Coordinate_out = '{}°{}°'.format(lat, lon)
            elif form_out == "LTM":
                Coordinate_out = '{}{}{}E{}N'.format(zone, h, E, N)
            elif form_out == "LPS":
                Coordinate_out = '{}{}E{}N'.format(h, E, N)
            elif form_out == "LGRS":
                Coordinate_out = '{}{}{}{}{}{}'.format(lonBand, latBand, e25k, n25k,
                                        E, N)
            elif form_out == "PolarLGRS":
                Coordinate_out = '{}{}{}{}{}'.format(lonBand, e25k, n25k, E, N)
            elif form_out == "LGRS_ACC":
                Coordinate_out = '{}{}{}{}{}{}{}{}'.format(lonBand, latBand,
                                        e25k, n25k, e1k, E, n1k, N)
            elif form_out == "PolarLGRS_ACC":
                Coordinate_out = '{}{}{}{}{}{}{}'.format(lonBand, e25k, n25k, e1k, E,
                                        n1k, N)
            elif form_out == "ACC" or form_out == "Polar_ACC":
                Coordinate_out = '{}{}{}{}'.format(e1k, E, n1k, N)
            else:
                raise ValueError("Operation aborted: output failed")
        else:
            if  form_out == "LatLon":
                Coordinate_out = '{} {}'.format(lat, lon)
            elif form_out == "LTM":
                Coordinate_out = '{} {} {} {}'.format(zone, h, E, N)
            elif form_out == "LPS":
                Coordinate_out = '{} {} {}'.format(h, E, N)
            elif form_out == "LGRS":
                Coordinate_out = '{} {} {} {} {} {}'.format(lonBand, latBand, e25k,
                                            n25k, E, N)
            elif form_out == "PolarLGRS":
                Coordinate_out = '{} {} {} {} {}'.format(lonBand, e25k, n25k, E, N)
            elif form_out == "LGRS_ACC":
                Coordinate_out = '{} {} {} {} {} {} {} {}'.format(lonBand, latBand,
                                    e25k, n25k, e1k, E, n1k, N)
            elif form_out == "PolarLGRS_ACC":
                Coordinate_out = '{} {} {} {} {} {} {}'.format(lonBand,
```

```
                                    e25k, n25k, e1k, E, n1k, N)
        elif form_out == "ACC" or form_out == "Polar_ACC":
            Coordinate_out = '{} {} {} {}'.format(e1k, E, n1k, N)
        else:
            raise ValueError("Operation aborted: output failed")

        print(Coordinate_out)

    except ValueError as e:
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        print(exc_type, fname, exc_tb.tb_lineno)
        print(e)
        exit(1)


# ----------------------------------------------------------------------
# Print program timing and elapsed time
    if info:
        EndTime = datetime.datetime.now()
        print('\n', ProgName, 'execution time')
        print('    Start Time:', StartTime)
        print('    End Time:  ', EndTime)
        print('    Run Time:  ', EndTime - StartTime)


# ======================================================================
# Run main program

# determine input argument format

info = False
trunc_val = 1
condensed = True

if info:
    StartTime = datetime.datetime.now()
    print('Program:', ProgName)

try:
    Method = sys.argv[1]
except IndexError:
    Method = None
    if info:
        print("No conversion method provided.")


# run conversion
try:
    if Method in ["LatLon2LTM",
                  "LatLon2LPS",
                  "LatLon2LGRS",
                  "LatLon2PolarLGRS",
                  "LatLon2LGRS_ACC",
                  "LatLon2PolarLGRS_ACC",
                  "LatLon2ACC",
```

```
            "LatLon2Polar_ACC",
            "LTM2LatLon",
            "LTM2LGRS",
            "LTM2LGRS_ACC",
            "LTM2ACC",
            "LPS2LatLon",
            "LPS2PolarLGRS",
            "LPS2PolarLGRS_ACC",
            "LPS2ACC",
            "LGRS2LTM",
            "PolarLGRS2LPS",
            "LGRS2LatLon",
            "PolarLGRS2LatLon",
            "LGRS_ACC2LatLon",
            "PolarLGRS_ACC2LatLon",
            "LGRS_ACC2LTM",
            "PolarLGRS_ACC2LPS",
            "LGRS2LGRS_ACC",
            "PolarLGRS2PolarLGRS_ACC",
            "LGRS2ACC",
            "PolarLGRS2Polar_ACC",
            "LGRS_ACC2LGRS",
            "PolarLGRS_ACC2PolarLGRS"]:

    initialize_LGRS_function_globals()
    main(Method, trunc_val, condensed)

else:
    raise ValueError("Operation aborted: Conversion Method Not Specified "
                     "Correctly.\n Select one of the following:\n"
                     "LatLon2LTM\n"
                     "LatLon2LPS\n"
                     "LatLon2LGRS\n"
                     "LatLon2PolarLGRS\n"
                     "LatLon2LGRS_ACC\n"
                     "LatLon2PolarLGRS_ACC\n"
                     "LatLon2ACC\n"
                     "LatLon2Polar_ACC\n"
                     "LTM2LatLon\n"
                     "LTM2LGRS\n"
                     "LTM2LGRS_ACC\n"
                     "LTM2ACC\n"
                     "LPS2LatLon\n"
                     "LPS2PolarLGRS\n"
                     "LPS2PolarLGRS_ACC\n"
                     "LPS2ACC\n"
                     "LGRS2LTM\n"
                     "LGRS2LatLon\n"
                     "LGRS_ACC2LTM\n"
                     "LGRS2LGRS_ACC\n"
                     "LGRS2ACC\n"
                     "PolarLGRS2LatLon\n"
                     "PolarLGRS2LPS\n"
                     "PolarLGRS2PolarLGRS_ACC\n"
                     "PolarLGRS2Polar_ACC\n"
                     "LGRS_ACC2LatLon\n"
```

```
                "LGRS_ACC2LGRS\n"
                "PolarLGRS_ACC2LatLon\n"
                "PolarLGRS_ACC2LPS\n"
                "PolarLGRS_ACC2\n")

except ValueError as e:
    if info:
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        print(exc_type, fname, exc_tb.tb_lineno)
        print(e)

        # ============================================================================
```

Appendix 2. Preliminary Lunar Grid Reference System Grid Generation Program

Foreword

This appendix contains preliminary code for a lunar grid generation program that was developed to generate the projected coordinate reference system (PCRS) and grids described in this documented. Grids, written as shapefiles, can be created for the Lunar Transverse Mercator (LTM) and Lunar Polar Stereographic (LPS) PCRS borders in both a planetocentric coordinate system and in their projections defined in the LPS and LTM sections. Grids in Lunar Grid Reference System (LGRS) and LGRS in Artemis Condensed Coordinates (ACC) formats of various sizes can also be generated with this software to the users desired resolution and in a specified location. Coordinate system conversions, when needed, are applied using the coordinate conversions preliminary software. The program operation is similar to the coordinate conversion code in appendix 1. Most combinations of grid resolution and coordinate system types are supported by this code and a grid can be written to a shapefile. However, not all grid resolutions are available, depending on the input or output format, such as a polar LGRS grid in the LTM portion of the grid system. As such, if the grid size and input and output combination are not allowed, the program should identify a grid that cannot be generated and terminate. Code operation and the Python code of the grid generation program are provided below.

Code Operation

The generic way to call the grid generation program is as follows:
./LGRS_Grid_Generation_mk7.py {Input format} {Output Format} {Grid resolution} {Location or (ULX ULY LRX LRY)}.
The input formats are as follows:

- LatLon—Latitude–longitude pair between latitudes −82° and 82°
- PolarLatLon—Polar latitude–longitude pair between latitudes −90° and −80° or latitudes 80° and 90°
- LTM—Lunar Transverse Mercator coordinates
- LPS—Lunar Polar Stereographic coordinates
- LGRS—Lunar Grid Reference System coordinates
- LGRS_ACC—Lunar Grid Reference System coordinates in Artemis Condensed Coordinates (ACC) format
- PolarLGRS—Lunar Grid Reference System coordinates in polar regions
- PolarLGRS_ACC—Lunar Grid Reference System coordinates in ACC format in polar regions

The output formats are the same as the input format and include two additional grids.

- LatLon
- PolarLatLon
- LTM
- LPS
- LGRS
- LGRS_ACC
- PolarLGRS
- PolarLGRS_ACC
- ACC—Lunar Grid Reference System coordinates in ACC format truncated to 6 characters.
- PolarACC—Lunar Grid Reference System coordinates in ACC format in polar regions truncated to 6 characters.

The output format only changes the way the grid areas are named; the grid positioning will be in LTM or LPS. The types of grids are

- global—Generate grid at global scale, LPS or LTM border LGRS global area reference (8° × 8°)
- global_all—Generate all global scale grids, all LTM borders, and LGRS global areas
- 25km_all—Generate plot of all 25-km-grid areas
- 25km—Generate 25-km grid within a single grid
- 1km—1-km grid
- 100m—100-m grid
- 10m—10-m grid
- 1m—1-m LGRS grid
- Multigrid—Plotting a grid across multiple zones in the LTM range of LGRS. The Multigrid option is not available in the LPS range.

For coarse grids such as 25 km and other larger grid sizes, the hemisphere and (or) zone is required. An example of a program call with sample input coordinates is as follows:

```
./LGRS_Grid_Generation_mk7.py LPS PolarLGRS global S
./LGRS_Grid_Generation_mk7.py LPS PolarLGRS 25km S
```

For grids finer than 25 km, a small box area needs to be provided to determine the correct area. One format to provide this information is

ULX ULY LRX LRY,

where

ULX is the upper left corner X coordinate value,
ULY is the upper left corner Y coordinate value,
LRX is the lower right corner X coordinate value, and
LRY is the lower right corner Y coordinate value.

A generic call for a LGRS grid in the LPS range is formatted as follows:

```
./LGRS_Grid_Generation_mk7.py LPS PolarLGRS 1km { ULX ULY LRX LRY }
```

Both the coordinate system and grid generation preliminary programs must be in the same directory to operate successfully: the grid conversion program needs to call the coordinate conversion program.

Please contact the U.S. Geological Survey or the Artemis Geospatial Data Team for support and assistance.

Preliminary LGRS Grid Generation Python Program

```
#!/bin/env python3
# ================================================================================
‘’’ PROGRAM INFORMATION
Program:  LGRS_Grid_Generation_mk7.py
Language: Python 3.11.4
Author:   Mark T. McClernan
Created:  March, 2024   (Python 3.11.4)
Modified: April, 2024   (Python 3.11.4)
IDE:      Visual Studio Code (v1.18.1 x86)

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
PROGRAM DESCRIPTION:
   This program generates Esri shapefiles to display the following
   lunar projected coordinate reference systems (PCRS) or grid systems:
      1. Lunar Transverse Mercator (LTM);
      2. Lunar Polar Stereographic (LPS);
```

```
    3. Lunar Grid Reference System (LGRS);
    4. LGRS in Artemis Condensed Coordinate Format (ACC)
and
    5. LGRS ACC coordinates limited to 6-characters.
Because of the condensed format of LGRS in ACC format, when specifying
input regions only 1 - 4 is able to be specified, LGRS in ACC format
assumes the knowledge of the LGRS 25km grid location. The grid generation
program will need this information to geospatially reference the correct
location.  Grids are generated by specifying an input and output coordinate
format and a bounding box that dimensions the grid area in formats 1 - 4.

Some grids only require that a portion in the grid area be specified, these
are generally small-scale and often cover a global resolution. For
simplicity, this software only creates shapefiles. If the bounding box
covers more than one area, the grid will be generated for both areas and
joined. For these areas covering multiple zones, the files will be
projected in LatLon regardless of which zones the grid area is plotted
in.

This software makes use of the LGRS_Coordinate_Conversion_mk7.py
program for all grid conversions and utilizes the OSGeo: GDAL, OGR,
and OGR Python packages for reading and writing shapefiles.

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
HOW TO USE THIS PROGRAM

This software is designed to run via the command line, where an input
and output format are provided with the grid size and input region or
coordinate box. The program will create an Esri shapefile or geopack
file with the grid.

The basic way to call the Grid Generation program is similar to
the Coordinate Conversion program in input structure with some
difference.

The program is called from the command line with the sample command:

./LGRS_Grid_Generation_mk7.py {Input format} {Output Format} \
    {Grid resolution} {Location or (ULX ULY LRX LRY)}

Only a specific combination of the input format, output format, and
the desired grid will produce an output file. Not all grid resolutions
are available, depending on the input or output format. The program has built
in warnings and checks that a correct output will only be generated
if the correct inputs are specified.

Program Required argument descriptions

{Input format}
    The accepted input formats are
    LatLon, LTM, LPS, LGRS, PolarLGRS, LGRS_ACC, and PolarLGRS_ACC.
    ACC or PolarACC cannot be specified on their own as these are relative
    coordinates.

    The input formats are:
    •       LatLon – LatLon pair between latitudes -82°<=lat<=+-82°
```

- PolarLatLon – polar LatLon pair between latitudes <-80° or >+80°
- LTM – Lunar Transverse Mercator
- LPS – Lunar Polar Stereographic
- LGRS – Lunar Grid Reference System
- LGRS_ACC - Lunar Grid Reference System in Artemis Condensed Format
- PolarLGRS - Lunar Grid Reference System in polar regions
- PolarLGRS_ACC - Lunar Grid Reference System in Artemis Condensed Format in polar regions. Specifying this format will change the way the input coordinates are read in and processed.

{Output Format}
Acceptable output coordinates are
LatLon, LTM, LPS, LGRS, PolarLGRS, LGRS_ACC, PolarLGRS_ACC, ACC or PolarACC.

Output formats are the same and include two additional grids.
- LatLon
- PolarLatLon
- LTM
- LPS
- LGRS
- LGRS_ACC
- PolarLGRS
- PolarLGRS_ACC
- ACC - Lunar Grid Reference System in Artemis Condensed Format truncated to 6 characters.
- PolarACC - Lunar Grid Reference System in Artemis Condensed Format in polar regions truncated to 6 characters.

The output format only changes the way the grid areas are named, the grid positioning will be in LTM or LPS.

{Input format} -> {Output Format}
This program does not provide the native ability to conduct inverse coordinate conversions like the LGRS_Coordonate_Conversion_mk* code. As such, only forward conversions from the input coordinate to the output coordinate will produce a grid shapefile.

List of acceptable I/O form in form out combinations:
- Form in -> Form Out

- -
- LatLon -> LTM, LGRS, LGRS_ACC, ACC
- PolarLatLon -> LPS, PolarLGRS, PolarLGRS_ACC, PolarACC
- LTM -> LTM, LGRS, LGRS_ACC, ACC
- LPS -> LPS, PolarLGRS, PolarLGRS_ACC, PolarACC
- LGRS -> LGRS, LGRS_ACC, ACC
- PolarLGRS -> PolarLGRS, PolarLGRS_ACC, PolarACC
- LGRS_ACC -> LGRS_ACC, ACC
- PolarLGRS_ACC -> PolarLGRS_ACC, PolarACC

- -
Inverse conversion should be completed with the Coordinate Conversion Program

{Grid resolution}
This program is capable of writing shapefiles for all grid resolutions of LTM, LPS, LGRS, and ACC formatted grids for the lunar equatorial and polar regions. There are some naming conventions.

Types of grids and descriptions
coarse grid { • global - Generate grid at global scale.
Used for LPS or LTM zone border in LPS/LTM coordinates
LGRS Global area reference (8°x8°) or (10°x8°) grid in LTM zone
• global_all - Generate all grids global scale grids if available
LTM zone borders and LGRS Global areas
All LGRS (8°x8°) or (10°x8°) global areas
LGRS Polar border, LPS border, for one hemisphere
• 25km - Generate 25km topocentric grid within a single zone
Generate LGRS 25km grid within one LTM zone
Generate Polar LGRS 25km grid within one LPS zone
• 25km_all - Generate plot of all 25km grid zones.
Generate all 25km grid areas within all LTM zones.
Generate all 25km grid areas within one LPS zone.
}
fine grid { • 1km - 1km LGRS grid in LGRS or ACC format
• 100m - 100m LGRS grid in LGRS or ACC format
• 10m - 10m LGRS grid in LGRS or ACC format
• 1m - 1m LGRS grid in LGRS or ACC format
}
{Location or (ULX ULY LRX LRY)}
Specifying the plotting region of the grid has two formats,

"Coarse" grids, defined as grid areas > 25km, only require the plotting region. This is either the LTM zone ("Zh") of the LPS Zone ("h").

"Fine" grids, defines as grids areas < 25km, require input of a bounding box to restrict the plotting region. To input these arguments, specific the upper left (UL) and lower right (LR) corners as follows ULX ULY LRX LRY, where
ULX - UL X or Easting
ULY - UL Y or Northing
LRX - LR X or Easting
LRY - LR Y or Northing
There arguments can be specified in planetocentric LatLon or topocentric Eastings and Northings. It was found to be easier to specify the grid area with topocentric coordinates during program development and testing.

For grid areas that cross multiple LTM zones, the user will be prompted whether they would like a multigrid. If yes, a grid will be created that crosses multiple zones. To support this operation, multigrid coordinates will only be exported in planetocentric LatLon. Program will terminate if a multigrid is not a desired output. This option is not needed for grids within a single LTM zone. Additionally, multigrids are not available to plot between the LTM and LPS portions of LPS, as the LPS and LTM portions are a notable boundary of LGRS and a multigrid is not really needed in polar regions. For data near the +-80° LPS/LGRS border, use the LTM extended range.

Example program calls
Coarse Grids
LatLon

```
        ./LGRS_Grid_Generation_mk7.py LatLon LTM border 30 45
        ./LGRS_Grid_Generation_mk7.py LatLon LTM border -30 -45

      PolarLatLon
        ./LGRS_Grid_Generation_mk7.py PolarLatLon LPS global 0 -85
        ./LGRS_Grid_Generation_mk7.py PolarLatLon LPS global 0 -85

      LTM
        ./LGRS_Grid_Generation_mk7.py LTM LTM border ${i} N

      LPS
        ./LGRS_Grid_Generation_mk7.py PolarLatLon LPS global 0 -85
        ./LGRS_Grid_Generation_mk7.py LPS LPS border N

      LGRS
        ./LGRS_Grid_Generation_mk7.py LTM LGRS global_all
        ./LGRS_Grid_Generation_mk7.py LTM LGRS 25km_all

        ./LGRS_Grid_Generation_mk7.py LTM LGRS global 23N
        ./LGRS_Grid_Generation_mk7.py LTM LGRS 25km 23N

      PolarLGRS
        ./LGRS_Grid_Generation_mk7.py LPS PolarLGRS global N
        ./LGRS_Grid_Generation_mk7.py LPS PolarLGRS global_all S

        ./LGRS_Grid_Generation_mk7.py LPS PolarLGRS 25km S
        ./LGRS_Grid_Generation_mk7.py LPS PolarLGRS 25km N

    fine grids
      equatorial LGRS_ACC (full name)
      ./LGRS_Grid_Generation_mk7.py LGRS_ACC LGRS_ACC 1km 30PED-000-000 30PFC-000-000
      ./LGRS_Grid_Generation_mk7.py LGRS_ACC LGRS_ACC 100m 30PED-000A000 30PEDA000-000
      ./LGRS_Grid_Generation_mk7.py LGRS_ACC LGRS_ACC 10m 30PED-000-100 30PED-100-000
      ./LGRS_Grid_Generation_mk7.py LGRS_ACC LGRS_ACC 1m 30PED-000-100 30PED-100-000

      polar PolarLGRS_ACC (full name)
      ./LGRS_Grid_Generation_mk7.py LPS PolarLGRS_ACC 1km S475000E600000N S525000E550000N
      ./LGRS_Grid_Generation_mk7.py LPS PolarLGRS_ACC 100m S475000E600000N S525000E550000N
      ./LGRS_Grid_Generation_mk7.py LPS PolarLGRS_ACC 10m S495000E575000N S505000E565000N
      ./LGRS_Grid_Generation_mk7.py LPS PolarLGRS_ACC 1m S495000E575000N S505000E565000N

    multigrid
      ./LGRS_Grid_Generation_mk7.py LatLon LGRS 1km 42.5 1.5 45.5 -1.5
      ./LGRS_Grid_Generation_mk7.py LatLon LGRS 100m 43.5 0.5 44.5 -0.5
      ./LGRS_Grid_Generation_mk7.py LatLon LGRS 10m 43.75 0.25 44.25 -0.25
      ./LGRS_Grid_Generation_mk7.py LatLon LGRS 1m 43.95 0.05 44.05 -0.05

  Artemis III Leibnitz Beta Plateau Nobile Rim 1
  Candidate Landing sites -  ACC grid example (truncated)

    Coarse grids
      ./LGRS_Grid_Generation_mk7.py LPS PolarLGRS global S
      ./LGRS_Grid_Generation_mk7.py LPS PolarLGRS 25km S
    fine grids
      ./LGRS_Grid_Generation_mk7.py PolarLatLon PolarACC 1km 27.288174340696408 -85.3270828326219
43.22853025996592 -85.47632419407181
```

```
            ./LGRS_Grid_Generation_mk7.py PolarLatLon PolarACC 100m 27.288174340696408 -85.3270828326219
43.22853025996592 -85.47632419407181
          WARNING: The following command produce extremely large files and take some time to process.
            ./LGRS_Grid_Generation_mk7.py PolarLatLon PolarACC 10m 27.288174340696408 -85.3270828326219
43.22853025996592 -85.47632419407181
            1m not available in ACC

    # - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    PROGRAM SECTIONS (can search for each using "SECTION #"):
      SECTION 0: INITIALIZE PROGRAM AND CALL EXTERNAL LIBRARIES
      SECTION 1: REFERENCE SURFACE PARAMETERS
        SECTION 1.1: LIST COORDINATE FORMATS
        SECTION 1.2: DEFINE MAP PROJECTIONS FOR PCRS AND GRID SYSTEMS USING
        SECTION 1.3: Define the LTM LPS coordinate limits and resolution
      SECTION 2: FUNCTION LIBRARY
        SECTION 2.1: Naming attribute table when creating shapefiles
        SECTION 2.2: Assign values to attribute table
        SECTION 2.3: Export multipolygon grid to Esri shapefile
        SECTION 2.4: Export single polygon, as a multipolygon data type, to
        SECTION 2.5: Assign X Y Coordinates to multipolygon osgeo ogr
        SECTION 2.6: Assign X Y positional Coordinates to single ogr ring
        SECTION 2.7: Assign X Y positional Coordinates to ogr rings in multi
              polygon geometry
        SECTION 2.8: Intersect ogr multipolygon geometries to a single geometry
        SECTION 2.9: Merge ogr multipolygon geometries to a single geometry
        SECTION 2.10: Clip multipolygon geometries within another geometry
        SECTION 2.11: Extract X, Y positions from multipolygon geometry
        SECTION 2.12: Calculate hemisphere location on a sphere in degrees
        SECTION 2.13: Calculate LTM zone prime meridian from a LTM zone number
        SECTION 2.14: Determine the LTM zone number from an input longitude.
        SECTION 2.15: Generate LPS Grid values
        SECTION 2.16: Generate LTM Grids values
        SECTION 2.17: Generate LGRS Grid values
        SECTION 2.18: Generate LGRS polar grid values
      SECTION 3: MAIN PROGRAM
        SECTION 3.1: determine if input arguments are correct
        SECTION 3.2: Read in input arguments center x center y or corners,
        SECTION 3.3: Converting Input Coordinates to Common Grid Format.
        SECTION 3.4: Generate the Correct Grid to the desired resolution
        SECTION 3.5: Convert grid Coordinates to correct output format.
        SECTION 3.6: Determine grid cell names in specified output format
        SECTION 3.7: Assign output WKT1 Projection string for writing file
        SECTION 3.8: Write grid to Esri shapefile and format output names

    ‘’’

    # ============================================================================
    # SECTION 0: INITIALIZE PROGRAM AND CALL EXTERNAL LIBRARIES


    import os               # used as system utilities
    import sys
                                # os -> Python 3.11.4
                                # sys 3.11.4

    import numpy as np      # To perform mathematical calculations
```

```
                    # and uses NumPy arrays for all grid
                    # positions
                            # np 1.25.2

import osgeo            # used to support geospatial data
                    # processing and formatting.

from osgeo import gdal,ogr,osr  # lib for rasters, vectors, and SRS
                      # Coordinate systems
                      # osgeo -> '3.7.1'
osr.DontUseExceptions() # suppress GDAL versioning warnings

from LGRS_Coordinate_Conversion_mk7 import *
                      # LGRS conversion lib for
                      # converting between
                      # LatLon, LTM, LPS, LGRS, LGRS_ACC
                # LGRS_Coordinate_Conversion_mk7 -> mk7

import re               # for input coordinate parsing
                    # re -> '2.2.1'

import datetime         # To perform time monitoring
                        # datetime -> Python 3.11.4

ProgName="LGRS_Grid_Generation_mk7"

# ============================================================================
# SECTION 1: REFERENCE SURFACE PARAMETERS
# Geodetic function constants for grid and grid generation on the Moon.
# Here we pre-assign the correct projection values and pre-dimension
# rectangular grids.

# ----------------------------------------------------------------------
# SECTION 1.1: LIST COORDINATE FORMATS


def initialize_LGRS_grid_generation_globals():
    ""
    Code creates globalized values to be called in the main program.
    Globals placed here, so constants can be easily changed in the
    future.

    Globals Defined:
    (list: str) coordinate_format: List of all the coordinate formats that
                    are able to be read into this program.
    (str) coordinate_format_str: List of input coordinate formats for
                    printing.
    (list: str) grid_format: List of all grid and grid resolutions that
                this program is capable of generating.
    (str) grid_format_str: List of grid and grid resolutions in a string
                for printing
    (re: str) LTM_pattern: re pattern used for reading in LTM coordinates
    (re: str) LTM_pattern_full: re pattern used for reading in LTM coordinates
            used at the grid corners that are condensed together
    (re: str) LGRS_pattern: re pattern used for reading condensed LGRS grid
            coordinates used at the grid corners.
```

```
    (re: str) PolarLGRS_pattern: re pattern used for reading condensed LGRS
                grid coordinates used at the grid corners on the
                lunar poles.
    (re: str) LGRS_ACC_pattern: re pattern used for reading LGRS
                    Artemis Condensed Coordinates (ACC) without
                    truncation. Coordinates are located at the
                    grid corners.
    (re: str) PolarLGRS_ACC_pattern: re pattern used for reading LGRS
                    Artemis Condensed Coordinates (ACC) without
                    truncation. Coordinates are located at the
                    grid corners on the lunar poles
    (re: str) LPS_pattern_full: re pattern used for reading in LPS
                        coordinates
    “””

    global coordinate_format, coordinate_format_str, grid_format
    global grid_format_str, LTM_pattern, LTM_pattern_full, LGRS_pattern
    global PolarLGRS_pattern, LGRS_ACC_pattern, PolarLGRS_ACC_pattern
    global LPS_pattern_full

    # input formats
    coordinate_format = [“LatLon”,”PolarLatLon”,”LTM”,”LPS”,”LGRS”,
                  “PolarLGRS”,”LGRS_ACC”,”PolarLGRS_ACC”]
    coordinate_format_str = “LatLon\nPolarLatLon\nLTM\nLPS\nLGRS\n\
                        PolarLGRS\nLGRS_ACC\nPolarLGRS_ACC\n\
                        ACC and PolarACC can be used for \n\
                        output Coordinates only.”

    # LGRS and 25km_all grid
    grid_format = [“border”,”global”,”global_all”,”25km”,”25km_all”,”1km”,”100m”,”10m”,”1m”]
    grid_format_str = “border\nglobal\nglobal_all\n25km\n25km_all\n1km\n100m\n10m\n1m”

    # supports floating or truncated integer values
    LTM_pattern = r’(\d+)([NS]{1})’
    LTM_pattern_full = r’(\d+)([NS]{1})(\d+)(E)(\d+)(N)’

    LPS_pattern_full = r’([NS]{1})(\d+(?:\.\d+)?|\.\d+)(E)(\d+(?:\.\d+)?|\.\d+)(N)’


    # supports only truncated values (this is ok; non-compressed
    # Coordinates can be read in as a full floating point)
    LGRS_pattern = r’(\d+)([A-Z]{1})([A-Z]{1})([A-Z]{1})(\d{5})(\d{5})’
    PolarLGRS_pattern = r’([A-Z]{1})([A-Z]{1})([A-Z]{1})(\d{5})(\d{5})’
    LGRS_ACC_pattern = r’(\d+)([A-Z]{1})([A-Z]{1})([A-Z]{1})([-A-Z]{1})(\d+)([-A-Z]{1})(\d+)’
    PolarLGRS_ACC_pattern = r’([A-Z]{1})([A-Z]{1})([A-Z]{1})([-A-Z]{1})(\d+)([-A-Z]{1})(\d+)’


# ----------------------------------------------------------------------
# SECTION 1.2: DEFINE MAP PROJECTIONS FOR PCRS AND GRID SYSTEMS USING
#             WKT1 STRINGS
    “””
    Define the the output map projections as WKT1 strings. Used when
    Exporting the files so they are georeferenced and display correctly
    on the lunar surface

    Globals Defined:
```

```
        (WKT1: str) N_LPS_projection_meters LPS northern zone, Defined by
                stereographic projection,
                projection center at 90°, 0°,
                Scale factor of 0.994
                False origins of 500km E, 500km N, and
                units in meters.
        (WKT1: str) S_LPS_projection_meters LPS Southern zone, Defined by
                stereographic projection,
                projection center at -90°, 0°,
                Scale factor of 0.994
                False origins of 500km E, 500km N, and
                units in meters.
        (WKT1: str) N_LTM_projection_meters LTM northern zones, Defined by
                transverse Mercator projection,
                projection center at 0° lam0 (longitude),
                Scale factor of 0.999
                False origins of 250km E, 2500km N, and
                units in meters. lam0 is unique in each zone.
        (WKT1: str) S_LTM_projection_meters LTM southern zones, Defined by
                transverse Mercator projection,
                projection center at 0° lam0,
                Scale factor of 0.999
                False origins of 250km E, 2500km N, and
                units in meters. lam0 is unique in each zone.
        (WKT1: str) Display_geocentric_degrees formats projection
                for the lunar spheroid with a semi major axis of
                1737400m with no associated flattening, f=0.
                Units are output in arc degrees.
                Projection may very based on default projection
                in GIS software.
        “”””
        global N_LPS_projection_meters,S_LPS_projection_meters
        global N_LTM_projection_meters,S_LTM_projection_meters
        global Display_geocentric_degrees

        # METERS <- LTM, LPS, LGRS, and ACC LGRS
        N_LPS_projection_meters = ‘’’PROJCS[“Moon (2015) - Sphere / Ocentric / North Polar”,
        GEOGCS[“Moon (2015) - Sphere / Ocentric”,DATUM[“Moon (2015) - Sphere”,SPHEROID[“Moon (2015)
- Sphere”,1737400,0,
        AUTHORITY[“IAU”,”30100”]],AUTHORITY[“IAU”,”30100”]],PRIMEM[“Reference Meridian”,0,AUTHORITY[“
IAU”,”30100”]],
        UNIT[“degree”,0.0174532925199433,AUTHORITY[“EPSG”,”9122”]],AUTHORITY[“IAU”,”30100”]],
        PROJECTION[“Polar_Stereographic”],
        PARAMETER[“latitude_of_origin”,90],
        PARAMETER[“central_meridian”,0],
        PARAMETER[“scale_factor”,0.994],
        PARAMETER[“false_easting”,500000],
        PARAMETER[“false_northing”,500000],
        UNIT[“metre”,1,AUTHORITY[“EPSG”,”9001”]],AXIS[“Easting”,EAST],AXIS[“Northing”,NORTH],
        AUTHORITY[“ARTEMIS”,”30197”]]’’’

        S_LPS_projection_meters = ‘’’PROJCS[“Moon (2015) - Sphere / Ocentric / South Polar”,
        GEOGCS[“Moon (2015) - Sphere / Ocentric”,DATUM[“Moon (2015) - Sphere”,SPHEROID[“Moon (2015)
- Sphere”,1737400,0,
        AUTHORITY[“IAU”,”30100”]],AUTHORITY[“IAU”,”30100”]],PRIMEM[“Reference Meridian”,0,AUTHORITY[“
IAU”,”30100”]],
```

```
        UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY["IAU","30100"]],
        PROJECTION["Polar_Stereographic"],
        PARAMETER["latitude_of_origin",-90],
        PARAMETER["central_meridian",0],
        PARAMETER["scale_factor",0.994],
        PARAMETER["false_easting",500000],
        PARAMETER["false_northing",500000],
        UNIT["metre",1,AUTHORITY["EPSG","9001"]],AXIS["Easting",EAST],AXIS["Northing",NORTH],
        AUTHORITY["ARTEMIS","30198"]]'"

        N_LTM_projection_meters  =  '"PROJCS["Moon (2015) - Sphere / Ocentric / Tranverse Mercator",
        GEOGCS["Moon (2015) - Sphere / Ocentric",DATUM["Moon (2015) - Sphere",SPHEROID["Moon (2015)
- Sphere",1737400,0,
        AUTHORITY["IAU","30100"]],AUTHORITY["IAU","30100"]],PRIMEM["Reference Meridian",0,AUTHORITY["
IAU","30100"]],
        UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY["IAU","30100"]],
        PROJECTION["Transverse_Mercator"],
        PARAMETER["latitude_of_origin",0],
        PARAMETER["central_meridian",{}],
        PARAMETER["scale_factor",0.999],
        PARAMETER["false_easting",250000],
        PARAMETER["false_northing",0],
        UNIT["metre",1,AUTHORITY["EPSG","9001"]],AXIS["Easting",EAST],AXIS["Northing",NORTH],
        AUTHORITY["ARTEMIS","30195"]]'"

        S_LTM_projection_meters = '"PROJCS["Moon (2015) - Sphere / Ocentric / Tranverse Mercator",
        GEOGCS["Moon (2015) - Sphere / Ocentric",DATUM["Moon (2015) - Sphere",SPHEROID["Moon (2015)
- Sphere",1737400,0,
        AUTHORITY["IAU","30100"]],AUTHORITY["IAU","30100"]],PRIMEM["Reference Meridian",0,AUTHORITY["
IAU","30100"]],
        UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY["IAU","30100"]],
        PROJECTION["Transverse_Mercator"],
        PARAMETER["latitude_of_origin",0],
        PARAMETER["central_meridian",{}],
        PARAMETER["scale_factor",0.999],
        PARAMETER["false_easting",250000],
        PARAMETER["false_northing",2500000],
        UNIT["metre",1,AUTHORITY["EPSG","9001"]],AXIS["Easting",EAST],AXIS["Northing",NORTH],
        AUTHORITY["ARTEMIS","30196"]]'"

        # DEGREES <- planetocentric latitude and longitude
        Display_geocentric_degrees = '"GEOGCS["Moon (2015) - Sphere / Ocentric",DATUM["Moon (2015) - Sphere",
        SPHEROID["Moon (2015) - Sphere",1737400,0,AUTHORITY["IAU","30100"]],AUTHORITY["IAU","30100"]],
        PRIMEM["Reference Meridian",0,AUTHORITY["IAU","30100"]],UNIT["degree",0.0174532925199433,
        AUTHORITY["EPSG","9122"]],AXIS["Latitude",NORTH],AXIS["Longitude",EAST],AUTHORITY["
IAU","30100"]]'"

    # ---------------------------------------------------------------------
    # SECTION 1.3: Define the LTM LPS coordinate limits and resolution
        ""
        Define the map projection and lunar spheroid constants to be used
        in dimensioning the grids.

        Globals Defined:
        (float) ResX grid plotting resolution in arc-degrees for X dimension
        (float) ResY grid plotting resolution in arc-degrees for Y dimension
```

```
(float, int) LTM_S_Border LTM southern projection boundary
(float, int) LTM_Equator planetocentric latitude
(float, int) LTM_N_Border LTM northern projection boundary
(float, int) LPS_S_Border most northern latitude for the S pole projection area
(float, int) LPS_N_Border most southern latitude for the N pole projection area
(float, int) LatLon_W_Meridian planetocentric min longitude
(float, int) LatLon_E_Meridian planetocentric max longitude
(float, int) LUNAR_RADIUS Lunar reference spheroid radial value
(float, int) LatLon_Central_Meridian Prime meridian of the moon
(float, int) LPS_S_phi0 LPS map projection center latitude value for S pole
(float, int) LPS_N_phi0 LPS map projection center latitude value for N pole
(float, int) LTM_Width LTM zone width of 8°
(float, int) FalseEasting LTM False Easting (Shift in X)
(float, int) FalseNorthing LTM False Northing (Shift in Y)
(float, int) FalseEasting_polar LPS False Easting (Shift in X)
(float, int) FalseNorthing_polar LPS False Northing (Shift in Y)
“”””

global ResX,ResY
global LTM_S_Border,LTM_Equator,LTM_N_Border,LPS_S_Border
global LPS_N_Border,LatLon_W_Meridian,LatLon_E_Meridian,LUNAR_RADIUS
global LatLon_Central_Meridian,LPS_S_phi0,LPS_N_phi0,LTM_Width
global FalseEasting,FalseNorthing,FalseEasting_polar,FalseNorthing_polar

# Define the shapefile point sampling frequency in arc-degrees.
# these arguments are not called for grids projected in rectangular
# Coordinates eg: LPS or LTM as the grid lines are vertical and
# horizontal in the right projection. This is used for plots that
# cover multiple zones
ResX = 0.01
ResY = 0.01

# define the LTM zone borders
LTM_S_Border = -82
LTM_Equator = 0
LTM_N_Border = 82
LTM_Width = 8
FalseEasting = 250E3
FalseNorthing = 2500E3

# LatLon borders
LatLon_W_Meridian = -180
LatLon_Central_Meridian = 0
LatLon_E_Meridian = 180
LUNAR_RADIUS = 1737400

#define the LPS zone borders
LPS_S_phi0 = -90
LPS_S_Border = -80
LPS_N_phi0 = 90
LPS_N_Border = 80
FalseEasting_polar = 500E3
FalseNorthing_polar = 500E3

# ========================================================================
```

```
# SECTION 2: FUNCTION LIBRARY

# SECTION 2.1: Naming attribute table when creating shapefiles
# SECTION 2.2: Assign values to attribute table
# SECTION 2.3: Export multipolygon grid to Esri shapefile
# SECTION 2.4: Export single polygon, as a multipolygon data type, to
# SECTION 2.5: Assign X Y Coordinates to multipolygon osgeo ogr
# SECTION 2.6: Assign X Y positional Coordinates to single ogr ring
# SECTION 2.7: Assign X Y positional Coordinates to ogr rings in multi
#              polygon geometry
# SECTION 2.8: Intersect ogr multipolygon geometries to a single geometry
# SECTION 2.9: Merge ogr multipolygon geometries to a single geometry
# SECTION 2.10: Clip multipolygon geometries within another geometry
# SECTION 2.11: Extract X, Y positions from multipolygon geometry
# SECTION 2.12: Calculate hemisphere location on a sphere in degrees
# SECTION 2.13: Calculate LTM zone prime meridian from a LTM zone number
# SECTION 2.14: Determine the LTM zone number from an input longitude.
# SECTION 2.15: Generate LPS Grid values
# SECTION 2.16: Generate LTM Grids values
# SECTION 2.17: Generate LGRS Grid values
# SECTION 2.18: Generate LGRS polar grid values

# ----------------------------------------------------------------------
# SECTION 2.1: Naming attribute table when creating shapefiles


def initialize_attribute_table(layer, attribute=True):
    “””
    Initialize the attribute table of an Esri shapefile.

    Input parameters:
    (str) Layer - osgeo osr layer definition
    (bool) attribute - switch to initialize the shapefile attribute table

    Defined Parameters:
    (int) id - polygon count - unitless
    (str) LTM | LPS | LGRS | PolarLGRS | LGRS_ACC | PolarLGRS_ACC | ACC |
    PolarACC - Grid coordinate designated at center of grid area
    (int) centX - center of grid square in Eastings -unit: deg or m
    (int) centY - center of grid square in Northings -unit: deg or m
    (int) ULX - upper left corner X -unit: arc-degrees or meters
    (int) ULY - upper left corner Y -unit: arc-degrees or meters
    (int) LRX - unit: lower right corner X -arc-degrees or meters
    (int) LRY - unit: lower right corner Y -arc-degrees or meters
    More can be added if needed

    Returns:
    (str) Layer - osgeo osr layer definition with attributes
                  initialized
    “””
    if attribute:
        # polygon count
        field_label = ogr.FieldDefn(‘id’, ogr.OFTInteger)
        layer.CreateField(field_label)

        # defined grid coordinate
```

```
        if form_out == “LTM”:
            field_label = ogr.FieldDefn(‘LTM’, ogr.OFTString)
            layer.CreateField(field_label)

        elif form_out == “LPS”:
            field_label = ogr.FieldDefn(‘LPS’, ogr.OFTString)
            layer.CreateField(field_label)

        elif ( ( form_out == “LGRS” ) or \
             ( form_out == “PolarLGRS” ) ):
            field_label = ogr.FieldDefn(‘LGRS’, ogr.OFTString)
            layer.CreateField(field_label)

        elif ( ( form_out == “LGRS_ACC” ) or \
              ( form_out == “PolarLGRS_ACC” ) ):
            field_label = ogr.FieldDefn(‘LGRS_ACC’, ogr.OFTString)
            layer.CreateField(field_label)

        elif ( ( form_out == “ACC” ) or \
              ( form_out == “PolarACC” ) ):

            field_label = ogr.FieldDefn(‘ACC’, ogr.OFTString)
            layer.CreateField(field_label)

        else:
            raise ValueError (“Coordinate could not be assigned”
                              “attribute table.”)

        # center Coordinates of the grid area
        field_label = ogr.FieldDefn(‘centX’, ogr.OFTInteger)
        layer.CreateField(field_label)

        field_label = ogr.FieldDefn(‘centY’, ogr.OFTInteger)
        layer.CreateField(field_label)

        # define corner points of the grid area
        field_label = ogr.FieldDefn(‘ULX’, ogr.OFTInteger)
        layer.CreateField(field_label)

        field_label = ogr.FieldDefn(‘ULY’, ogr.OFTInteger)
        layer.CreateField(field_label)

        field_label = ogr.FieldDefn(‘LRX’, ogr.OFTInteger)
        layer.CreateField(field_label)

        field_label = ogr.FieldDefn(‘LRY’, ogr.OFTInteger)
        layer.CreateField(field_label)

    return layer

# ----------------------------------------------------------------------
# SECTION 2.2: Assign values to attribute table


def set_field_values(feat, k, coord, X, Y, attribute):
    “””
```

```
Initialize the attribute table of an Esri shapefile.

Input Parameters:
(str) feat - osgeo osr polygon feature, part of a layer
(int) k - index referenced to a value assigned to a specific polygon
        in the total polygon count
(bytes) coord - Grid area name, truncated to specified value
(array; doubles) X - Grid X or easting values -units:m
(array; doubles) Y - Grid Y or northing values -units:m
(bool) attribute - switch to initialize the shapefile attribute table

Assigned Parameters:
(int) id - polygon count - unitless
(str) LTM | LPS | LGRS | PolarLGRS | LGRS_ACC | PolarLGRS_ACC | ACC |
PolarACC - Grid coordinate designated at center of grid area
(int) centX - center of grid square in Eastings -unit: deg or m
(int) centY - center of grid square in Northings -unit: deg or m
(int) ULX - upper left corner X -unit: arc-degrees or meters
(int) ULY - upper left corner Y -unit: arc-degrees or meters
(int) LRX - unit: lower right corner X -arc-degrees or meters
(int) LRY - unit: lower right corner Y -arc-degrees or meters
More can be added if needed

Returns:
(str) Layer - osgeo osr layer definition with attributes
            initialized
“””
if attribute:
    # set unique number for each polygon
    feat.SetField(‘id’, k)

    # assign coordinate value for the output format to table
    if form_out == “LTM”:
        feat.SetField(‘LTM’, coord)

    elif form_out == “LPS”:
        feat.SetField(‘LPS’, coord)

    elif ( ( form_out == “LGRS” ) or \
           ( form_out == “PolarLGRS” ) ):
        feat.SetField(‘LGRS’, coord)

    elif ( ( form_out == “LGRS_ACC” ) or \
           ( form_out == “PolarLGRS_ACC” ) ):
        feat.SetField(‘LGRS_ACC’, coord)

    elif ( ( form_out == “ACC” ) or \
           ( form_out == “PolarACC” ) ):
        feat.SetField(‘ACC’, coord)

    else:
        raise ValueError (“Coordinate could not be assigned: “
                          “attribute table.”)

    # calculate grid area extrema
    n = np.nanmax(Y)
```

```
        s = np.nanmin(Y)
        e = np.nanmax(X)
        w = np.nanmin(X)

        # calculate center coordinate value
        CentX = (w + e)/2
        CentY = (s + n)/2

        # assign center coordinate value to table
        feat.SetField('centX', round(CentX))
        feat.SetField('centY', round(CentY))

        # assign grid corner coordinates to table
        feat.SetField('ULX', round(w))
        feat.SetField('ULY', round(n))
        feat.SetField('LRX', round(e))
        feat.SetField('LRY', round(s))

        return feat

# ------------------------------------------------------------------------
# SECTION 2.3: Export multipolygon grid to Esri shapefile


def Xport_multipolygons(X, Y, GridName, f_o,
                        WKT1,
                        attribute=True,
                        output_file_form="shp",
                        overwrite=False):
    """
    This section of code reads X Y coordinates of degrees of meters and
    writes them to an Esri shapefile. This code writes out a multipolygon
    geometry, by writing multiple osgeo ogr rings. it cannot out
    put a single polygon layer. Writing of the attribute table is
    optional. Additionally, the projection is assigned using a WKT1
    string, and can be changed to a more custom projection if needed.
    Map projection WKT1 strings are assigned at the beginning of this
    program.

    Input Parameters:
    (array; doubles) X - Grid X or easting values -units:m
    (array; doubles) Y - Grid Y or northing values -units:m
    (bytes) GridName - Grid area name, truncated to specified value
    (str) f_o - output file name and output folder
    (str) WKT1 - Well-Known Text 1 formatted string with output projection
    (bool) attribute - switch to initialize the shapefile attribute table
    (str) output_file_form - specify file output format
                        - sph for esri shapefile
                        - gpkg for geopackage
    (bool) overwrite - switch to initialize that a shapefile be
                    overwritten if it already exists

    Returns:
    (file) - Esri shapefiles (shp,shx,dbf,prj) with name of f_o
          - gpgk and other files are also written if a different
          - file format is specified
```

```
(folder) - Esri shapefiles containing folder with name of f_o
“””
# Procedure to remove a shapefile if it exists to its
# file name can be reused when rewriting a new shapefile

if output_file_form == “shp”:
   if overwrite:
      if os.path.isfile(f_o + “/” + f_o + “.shp”):
         print(“file:”,f_o + “/” + f_o + “.shp”,”exists removing and rewriting”)
         os.remove(f_o + “/” + f_o + “.shp”)
         os.remove(f_o + “/” + f_o + “.shx”)
         os.remove(f_o + “/” + f_o + “.dbf”)
         os.remove(f_o + “/” + f_o + “.prj”)
   else:
      # If overwriting is not specified, then the program terminates
      if os.path.isfile(f_o + “/” + f_o + “.shp”):
         print(“file:”,f_o + “/” + f_o + “.shp”,”exists terminating.”)
         raise ValueError (“Terminating File cannot be overwritten. “)
         exit()

   # initialize shapefile for exporting
   driver = ogr.GetDriverByName(‘ESRI Shapefile’)
   ds  =  driver.CreateDataSource(f_o)
   srs = osr.SpatialReference()
   srs.ImportFromWkt(WKT1)
   layer = ds.CreateLayer(f_o, srs, geom_type=ogr.wkbMultiPolygon)

# format for a geopackage file type
elif output_file_form == “gpkg”:
   if overwrite:
      if os.path.isfile(f_o + “.gpkg”):
         print(“file:”,f_o + “.gpkg”,”exists removing and rewriting”)
         os.remove(f_o + “.gpkg”)

   else:
      # If overwriting is not specified, then the program terminates
      if os.path.isfile(f_o + “.gpkg”):
         print(“file:”, f_o + “.gpkg”,”exists terminating.”)
         raise ValueError (“Terminating File cannot be overwritten. “)
         exit()

   driver=ogr.GetDriverByName(‘GPKG’)
   ds = driver.CreateDataSource(f_o+”.gpkg”)
   srs=osr.SpatialReference()
   srs.ImportFromWkt(WKT1) # assigning projection to file
   layer = ds.CreateLayer(f_o,srs,geom_type=ogr.wkbMultiPolygon)

# Catch all if the output format could not be determined
else:
   raise ValueError(“Cannot write output file: Output format could “
                    “not be determined”)

# Geopackage as example if you want to support an alternative format
#driver=ogr.GetDriverByName(‘GPKG’)
#ds = driver.CreateDataSource(f_o+”.gpkg”)
#srs=osr.SpatialReference()
```

```
    #srs.ImportFromWkt(WKT1) # assigning projection to file
    #layer = ds.CreateLayer(f_o,srs,geom_type=ogr.wkbMultiPolygon)

    # set up attributes layer from external function
    layer = initialize_attribute_table(layer,attribute)

    # get layer feature definition
    feature_defn = layer.GetLayerDefn()

    # loop through grid and assign grid areas to geometry
    k = 0
    for i in range(0, X.shape[0]):   # change line
       PolygonString = ogr.Geometry(ogr.wkbPolygon)
       ring = ogr.Geometry(ogr.wkbLinearRing)

       if np.isnan(X[i,0]): # skip if empty ring
          continue

       for j in range(0, X.shape[1]): # iterate though line lines
          if not np.isnan(X[i,j]): # if data value exits continue
                            # nan is used to end the grid
             ring.AddPoint(X[i,j], Y[i,j])
       ring.CloseRings()

       # add geometry to string
       PolygonString.AddGeometry(ring)

       # add polygon to multi polygon
       MultiPolygonString = ogr.Geometry(ogr.wkbMultiPolygon)
       MultiPolygonString.AddGeometry(PolygonString)

       # assign attributes to shapefile if specified
       feat = ogr.Feature(feature_defn)
       if attribute:
          # function to assign attribute values using
          # an external function
          feat=set_field_values(feat,
                        k,
                        GridName[i,0].decode(),
                        X[i,:],
                        Y[i,:],
                        attribute)

       # assign polygons to feature
       feat.SetGeometry(MultiPolygonString)
       layer.CreateFeature(feat)

       # iterate id value for next shapefile
       k += 1

       # clean up memory and continue for next grid area to be
       # dimensioned
       feat = None
    ds = layer = feat = None

# ----------------------------------------------------------------------
```

```
# SECTION 2.4: Export single polygon, as a multi polygon data type, to
#              an Esri shapefile.

def Xport_polygon(X, Y, GridName, f_o,
                  WKT1,
                  attribute=True,
                  output_file_form="shp",
                  overwrite=False):
    """
    This section of code reads X Y coordinates of degrees of meters and
    writes them to a shapefile or geopack. This code writes out a multipolygon
    geometry for a single polygon ogr ring. Writing of the attribute
    table is optional. Additionally, the projection is assigned using
    a WKT1 string, and can be changed to a more custom projection if
    needed. Map projection WKT1 strings are assigned at the beginning of
    this program.

    Input Parameters:
    (array; doubles) X - Grid X or easting values -units:m
    (array; doubles) Y - Grid Y or northing values -units:m
    (bytes) GridName - Grid area name, truncated to specified value
    (str) f_o - output file name and output folder
    (str) WKT1 - WKT1 formatted string with output projection
    (bool) attribute - switch to initialize the shapefile attribute table
    (str) output_file_form - specify file output format
                           - sph for Esri shapefile
                           - gpkg for geopackage
    (bool) overwrite - switch to initialize that a shapefile be
                       overwritten if it already exists

    Returns:
    (file) - Esri shapefiles (shp,shx,dbf,prj) with name of f_o
           - gpgk and other files are also written if a different
           - file format is specified
    (folder) - Esri shapefiles containing folder with name of f_o
    """

    # Procedure to remove a shapefile if it exists
    # file name can be reused when rewriting a new shapefile
    if output_file_form == "shp":
        if overwrite:
            if os.path.isfile(f_o + "/" + f_o + ".shp"):
                print("file:",f_o + "/" + f_o + ".shp","exists removing and rewriting")
                os.remove(f_o + "/" + f_o + ".shp")
                os.remove(f_o + "/" + f_o + ".shx")
                os.remove(f_o + "/" + f_o + ".dbf")
                os.remove(f_o + "/" + f_o + ".prj")
        else:
            # If overwriting is not specified, then the program terminates
            if os.path.isfile(f_o + "/" + f_o + ".shp"):
                print("file:",f_o + "/" + f_o + ".shp","exists terminating.")
                raise ValueError ("Terminating File cannot be overwritten. ")
                exit()

        # initialize shapefile for exporting
        driver = ogr.GetDriverByName('ESRI Shapefile')
```

```
    ds  =  driver.CreateDataSource(f_o)
    srs = osr.SpatialReference()
    srs.ImportFromWkt(WKT1)
    layer = ds.CreateLayer(f_o, srs, geom_type=ogr.wkbMultiPolygon)

# format for a geopackage file type
elif output_file_form == "gpkg":
    if overwrite:
        if os.path.isfile(f_o + ".gpkg"):
            print("file:",f_o + ".gpkg","exists removing and rewriting")
            os.remove(f_o + ".gpkg")

    else:
        # If overwriting is not specified, then the program terminates
        if os.path.isfile(f_o + "/" + f_o + ".gpkg"):
            print("file:",f_o + "/" + f_o + ".gpkg","exists terminating.")
            raise ValueError ("Terminating File cannot be overwritten. ")
            exit()

    driver=ogr.GetDriverByName('GPKG')
    ds = driver.CreateDataSource(f_o+".gpkg")
    srs=osr.SpatialReference()
    srs.ImportFromWkt(WKT1) # assigning projection to file
    layer = ds.CreateLayer(f_o,srs,geom_type=ogr.wkbMultiPolygon)

# Catch all if the output format could not be determined
else:
    raise ValueError("Cannot write output file: Output format could "
                     "not be determined")

# set up attributes layer
layer = initialize_attribute_table(layer,attribute)

# get layer feature definition
feature_defn = layer.GetLayerDefn()

# Create polygon
PolygonString = ogr.Geometry(ogr.wkbPolygon)
ring = ogr.Geometry(ogr.wkbLinearRing)

# assign attributes to shapefile if specified
for i in range(0, X.shape[0]):   # change grid area
    for j in range(0, X.shape[1]):   # iterate polygon area
        ring.AddPoint(X[i,j], Y[i,j])
ring.CloseRings()

# add geometry to polygon
PolygonString.AddGeometry(ring)

# assign polygons to feature
MultiPolygonString = ogr.Geometry(ogr.wkbMultiPolygon)
MultiPolygonString.AddGeometry(PolygonString)

k = 1 # assign id value

# assign values to attribute table if specified
```

```
    feat = ogr.Feature(feature_defn)
    if attribute:
        # function to assign attribute values using
        # an external function
        feat = set_field_values(feat,
                                k,
                                GridName[0,0].decode(),
                                X[0,:],
                                Y[0,:],
                                attribute)
    # assign polygons to feature
    feat.SetGeometry(MultiPolygonString)
    layer.CreateFeature(feat)

    # clean up memory and end function
    ds = layer = feat = None

# ----------------------------------------------------------------------
# SECTION 2.5: Assign X Y Coordinates to multi polygon osgeo ogr
#              geometry.


def assign_xy_grid_to_multipolygon(X, Y):
    """
    This section of code reads X Y coordinates of degrees of meters and
    writes them to an osgeo ogr multiplication geometry. Map projection
    does not matter, however, please note, any clipping or additional
    operations will need to be completed in the exact same coordinate
    format to avoid issues.This call only works with grid formatted
    areas, thus a bounding box area will not work

    Input Parameters:
    (2D array; doubles) X - Grid X or easting values -units:m
    (2D array; doubles) Y - Grid Y or northing values -units:m

    Returns:
    (class) - MultiPolygonString multipolygon string of a grid area
    """
    # create new multipolygon geometry
    MultiPolygonString = ogr.Geometry(ogr.wkbMultiPolygon)

    # Assign X and Y values to geometry, 1 in removed from the total
    # looping size as this is plotted like a grid, and all four corner
    # points are assigned at the same time
    for i in range(0,X.shape[0]-1): # iterate through grid areas
        for j in range(0,X.shape[1]-1): # iterate through grid bounding area

            # create geometry to store location information
            PolygonString = ogr.Geometry(ogr.wkbPolygon)
            ring = ogr.Geometry(ogr.wkbLinearRing)

            # add all four corner points of the grid area to ring
            ring.AddPoint(X[i,j], Y[i,j])
            ring.AddPoint(X[i,j+1], Y[i,j+1])
            ring.AddPoint(X[i+1,j+1], Y[i+1,j+1])
            ring.AddPoint(X[i+1,j], Y[i+1,j])
```

```
            ring.AddPoint(X[i,j], Y[i,j])
            ring.CloseRings() # close ring

            # add geometry to single polygon
            PolygonString.AddGeometry(ring)

            # add polygon to multipolygon
            MultiPolygonString.AddGeometry(PolygonString)

    # export multipolygon
    return MultiPolygonString

# ----------------------------------------------------------------------
# SECTION 2.6: Assign X Y positional Coordinates to single ogr ring
#              geometry

def assign_xy_to_polygon (X, Y):
    “””
    This section of code reads X Y coordinates of degrees of meters and
    writes them to an osgeo ogr polygon geometry. Map projection
    does not matter, however, please note, any clipping or additional
    operations will need to be completed in the exact name coordinate
    format to avoid issues. This call only works with a single polygon,
    however, the polygon geometry can be longer that the 5 values LGRS
    grids are written in. For grids see the
    assign_xy_grid_to_multipolygon function.

    Input Parameters:
    (2D array; doubles) X - Grid X or easting values -units:m
    (2D array; doubles) Y - Grid Y or northing values -units:m

    Returns:
    (class) - PolygonString single polygon geometry string of a grid
              area
    “””
    # create new polygon geometry and ring to assign location values
    PolygonString = ogr.Geometry(ogr.wkbPolygon)
    RingString = ogr.Geometry(ogr.wkbLinearRing)

    # Assign X and Y values to ring
    for i in range(0, X.shape[0]): # loop through polygons
        for j in range(0, X.shape[1]): # loop through geometry
            if np.isnan(X[i,j]):
                continue
            RingString.AddPoint(X[i,j], Y[i,j]) # assign value

    # finalize polygon geometry
    RingString.CloseRings()

    # add ring geometry to polygon
    PolygonString.AddGeometry(RingString)

    # export finalized polygon
    return PolygonString

# ----------------------------------------------------------------------
```

```
# SECTION 2.7: Assign X Y positional Coordinates to ogr rings in multi
#              polygon geometry

def assign_arbitrary_xy_to_multipolygon (X, Y):
    """
    This section of code reads X Y coordinates of degrees of meters and
    writes them to an osgeo ogr multipolygon geometry. Map projection
    does not matter, however, please note, any clipping or additional
    operations will need to be completed in the exact same coordinate
    format to avoid issues.For grids see the
    assign_xy_grid_to_multipolygon function. For single polygons see
    the assign_xy_to_polygon function.

    Input Parameters:
    (2D array; doubles) X - Grid X or easting values -units:m
    (2D array; doubles) Y - Grid Y or northing values -units:m

    Returns:
    (class) - PolygonString single polygon geometry string of a grid
            area
    """
    # create new polygon geometry and ring to assign location values to
    # Create a new multipolygon geometry
    MultiPolygon = ogr.Geometry(ogr.wkbMultiPolygon)

    # Iterate through each polygon
    for i in range(X.shape[0]):  # Loop through polygons
        if np.isnan(X[i, 0]):
            continue
        # Create a new polygon geometry and ring
        PolygonString = ogr.Geometry(ogr.wkbPolygon)
        RingString = ogr.Geometry(ogr.wkbLinearRing)

        for j in range(X.shape[1]):  # Loop through points in the polygon
            if np.isnan(X[i, j]):
                continue
            RingString.AddPoint(X[i, j], Y[i, j])  # Add point to ring

        # Close the ring
        RingString.CloseRings()

        # Add ring geometry to polygon
        PolygonString.AddGeometry(RingString)

        # Add the polygon to the multipolygon
        MultiPolygon.AddGeometry(PolygonString)

    return MultiPolygon

# ----------------------------------------------------------------------
# SECTION 2.8: Intersect ogr multipolygon geometries to a single geometry

def intersect_multipolygons(multipolygon1, multipolygon2):
    """Intersect two MULTIPOLYGON Z geometries and return the result."""
    # Create an empty multipolygon to store the result
    intersect_multipolygon = ogr.Geometry(ogr.wkbMultiPolygon)
```

```
    # Iterate over each polygon in the first multipolygon
    for i in range(multipolygon1.GetGeometryCount()):
        polygon1 = multipolygon1.GetGeometryRef(i)
        # Iterate over each polygon in the second multipolygon
        for j in range(multipolygon2.GetGeometryCount()):
            polygon2 = multipolygon2.GetGeometryRef(j)

            # Perform intersection between the two polygons
            intersected_polygon = polygon1.Intersection(polygon2)
            # Add the intersected polygon to the result if it's not empty
            if not intersected_polygon.IsEmpty():
                intersect_multipolygon.AddGeometry(intersected_polygon)

    return intersect_multipolygon

# ----------------------------------------------------------------------
# SECTION 2.9: Merge ogr multipolygon geometries to a single geometry

def merge_multipolygons(geometries1, geometries2):
    """
    The merge_multipolygons combine the two geometries of polygon or
    multipolygon format into a single multipolygon format. The
    function recursively merges multipolygons within each polygon
    geometry, so every polygon is re assigned to a new geometry.
    It checks if the geometry is a multipolygon, and if so, recursively
    constructs new polygons. This process continues until all levels of
    multipolygons are merged into a single ogr geometry object
    representing the merged multipolygon. Map projection or the Coordinates
    does not matter, however, please note, any clipping or additional
    operations will need to be completed in the exact same coordinate
    format to avoid issues.

    Input Parameters:
    (class) geometries1 - set of geometry values formatted as an ogr multi or
                single polygon
    (class) geometries2 - set of geometry values formatted as an ogr multi or
                single polygon

    Returns:
    (class) merged_multipolygon - multipolygon composed of both geometries1
                    and geometries2.
    """
    # create new multipolygon geometry
    merged_multipolygon = ogr.Geometry(ogr.wkbMultiPolygon)

    # Iterate over geometries from the first set
    for geometry in geometries1:
        # Check if the geometry is a multipolygon
        if geometry.GetGeometryType() == ogr.wkbMultiPolygon:
            # If it's a multipolygon, recursively merge its constituent polygons
            merged_multipolygon.AddGeometry(merge_multipolygons(geometry, []))
        else:
            # If it's a polygon, add it directly to the merged multipolygon
            merged_multipolygon.AddGeometry(geometry)
```

```
    # Iterate over geometries from the second set
    for geometry in geometries2:
        # Check if the geometry is a multipolygon
        if geometry.GetGeometryType() == ogr.wkbMultiPolygon:
            # If it's a multipolygon, recursively merge its constituent polygons
            merged_multipolygon.AddGeometry(merge_multipolygons(geometry, []))
        else:
            # If it's a polygon, add it directly to the merged multipolygon
            merged_multipolygon.AddGeometry(geometry)

    # return new multipolygon of merged areas
    return merged_multipolygon

# ----------------------------------------------------------------------
# SECTION 2.10: Clip multipolygon geometries within another geometry


def clip_multipolygon_by_polygon(multipolygon_geometry, clip_polygon):
    """
    The clip_multipolygon_by_polygon takes two geometries, the
    multipolygon_geometry (ogr multipolygon) and the clip_polygon
    (ogr polygon) in order to clip the multipolygon_geometry to be
    contained within the clip_geometry. This function has the capability
    of removing polygon areas, or cutting them to ensure that if a
    polygon has a portion of its area in the clip geometry is retained
    in the final polygon.

    Input Parameters:
    (class) multipolygon_geometry - set of geometry values formatted as
                an ogr multi or single polygon
    (class) clip_polygon -set of geometry values formatted as an ogr
                multi or single polygon

    Returns:
    (class) merged_multipolygon - multipolygon composed of
                multipolygon_geometry within the region of
                clip_polygon.
    """
    # Create a new empty geometry to store the clipped result
    clipped_geometry = ogr.Geometry(ogr.wkbMultiPolygon)

    # Iterate over each polygon in the multipolygon geometry
    for i in range(multipolygon_geometry.GetGeometryCount()):
        polygon = multipolygon_geometry.GetGeometryRef(i)

        # Perform intersection with the clip polygon
        clipped_polygon = polygon.Intersection(clip_polygon)

        # Add the clipped polygon to the result if it's not empty
        if not clipped_polygon.IsEmpty():
            clipped_geometry.AddGeometry(clipped_polygon)

    return clipped_geometry

# ----------------------------------------------------------------------
# SECTION 2.11: Extract X, Y potions from multipolygon geometry
```

```
def extract_xy_from_multipolygon(multipolygon_geometry):
    “””
    extract_xy_from_multipolygon takes a multipolygon input geometry,
    and extracts the X and Y position and returns them as a NumPy array.

    Input Parameters:
    (class) multipolygon_geometry - set of geometry values formatted as an ogr multi or
                    single polygon

    Returns:
    (class) merged_multipolygon - multipolygon composed of
                multipolygon_geometry within the region of
                clip_polygon.
    “””
    # Initialize lists to store X and Y coordinates

    # calculate the max number of points in a polygon, this is used
    # to dimension the arrays and to save memory
    max_points = 0
    for i in range(multipolygon_geometry.GetGeometryCount()):
        polygon = multipolygon_geometry.GetGeometryRef(i)
        ring = polygon.GetGeometryRef(0)
        points2 = ring.GetPointCount()
        if points2 > max_points:
            max_points = points2

    # Dimension arrays by number of polygons and max number of points
    # in polygon
    x_values = np.full((multipolygon_geometry.GetGeometryCount(), max_points), np.nan)
    y_values = np.full((multipolygon_geometry.GetGeometryCount(), max_points), np.nan)

    # Iterate over each polygon in the multipolygon and
    # assign geometry X, Y to arrays
    k = 1  # keep track of total polygon count
    for i in range(multipolygon_geometry.GetGeometryCount()):
        polygon = multipolygon_geometry.GetGeometryRef(i)
        ring = polygon.GetGeometryRef(0)
        points = ring.GetPointCount()

        # Extract X and Y coordinates from the polygon and assign
        # to NumPy array
        for j in range(0,points):
            x, y, _ = ring.GetPoint(j)
            x_values[i,j] = x
            y_values[i,j] = y

        k+=1 # iterate k

    return x_values, y_values

# ----------------------------------------------------------------------
# SECTION 2.12: Calculate hemisphere location on a sphere in degrees
#               Return numeric value
```

```
def Calc_hemisphere(phi):
    “””
    determine the correct lunar hemisphere given a position in latitude

    Input Parameters:
    (float, int) phi - planetocentric latitude in the range of
                    -90° - 90°.

    Returns:
    (str) hemisphere - return string value of the hemisphere location
                        either “N” for northern or “S” for southern.
    “””
    if phi >= 0: # We assume the equator to be northern
        return “N”
    elif phi < 0:
        return “S”
    else:
        raise ValueError(“Hemisphere could not be determined”)

# ----------------------------------------------------------------------
# SECTION 2.13: Calculate LTM zone prime meridian from a LTM zone number


def Calc_LTM_prime_meridian(zone, type):
    “””
    Determine the correct LTM central meridian planetocentric longitude
    given an LTM number in the range of [1-45]. Central meridian can be
    returned in either radian of arc-degrees

    Input Parameters:
    (int) zone - LTM 8° W zone number. Northern and Southern zones and
                 their difference not utilized.
    (str) typ - the type of output, radians or degrees, needed from
                this calculation

    Returns:
    (float) lam0 - longitude of LTM zone central meridian.
    “””
    # safety checks
    if zone > 45:
        raise ValueError (“LTM zone 46\{N|S\} of higher does not exist”)
    elif zone < 0:
        raise ValueError (“Negative LTM zones do not exist”)

    # determine the type of calculation required
    if type == “rad”:
        lam0 = np.deg2rad((zone-1) * (LTM_Width) - 180. + (LTM_Width/2))
    elif type == “deg”:
        lam0 = (zone-1) * (LTM_Width) - 180. + (LTM_Width/2)
    else:
        raise ValueError (“LTM zone prime meridian could not be “
                          “calculated: place specify type “
                          “\{\”rad\”|\”deg\”\}”)

    return lam0
```

```
# ----------------------------------------------------------------------
# SECTION 2.14: Determine the LTM zone number from an input longitude.


def Calc_LTM_zone(X):
    “””
    Determine the correct LTM zone number from an input longitude.

    Input Parameters:
    (int, float) X - LTM formatted in plenetocentric -180 to 180 degrees.
            Northern and Southern zones and their difference not
            utilized.

    Returns:
    (int) zone - LTM zone number without hemisphere designation.
    “””
    zone = np.floor((np.mean(X) + 180)/(LTM_Width)) + 1
    return int(zone)

# ----------------------------------------------------------------------
# SECTION 2.15: Generate LPS Grid values according to a specified
#             resolution and coordinate type


def generate_LPS_grid(Grid_Res, h):
    “””
    This section of code creates all Lunar Polar Stereographic (LPS)
    systems grid given a small number of inputs.

    Input Parameters:
    (str) Grid_Res - Grid format and resolution
    (str) h - hemisphere (either north, “N” | south, “S”)

    Valid types of grids
    Grid_Res == border - Single polygon of the LPS 80° border plotted
                    plotted in Latitude and Longitude coordinates.
                    This grid is for display in a global context
                    as box is drawn around the plot area.
    Grid_Res == global - Single polygon of the LPS 80° border plotted
                    in LPS Coordinates. Data is exported in the
                    LPS projections depending on the zone
                    location. This plot can only be plotted in
                    polar stereographic. No points are placed
                    at the projection centers (90° or-90°) and
                    will display as a line if in the wrong
                    projection.
    Returns:
    (2D array, floats) GridX - Longitude or Easting values of the LPS
                        border.
    (2D array, floats) GridY - Latitude or Northing values of the LPS
                        border.
    “””
    # plot LPS areas for display in LatLon. This plots a box
    # in latitude and longitude values.
    if Grid_Res == “border”:
        # Southern zone
```

```
    if h == "S":
        # grid-lines W->E
        GridX_T = np.arange(LatLon_W_Meridian,LatLon_E_Meridian,ResX)[:,np.newaxis]
        GridX_T = np.append(GridX_T,LatLon_E_Meridian)

        GridY_T = np.ones(GridX_T.shape[0]-1).T*LPS_S_Border
        GridY_T = np.append(GridY_T,LPS_S_Border) #S

        # draw E->W values
        GridX_B = np.arange(LatLon_E_Meridian,LatLon_W_Meridian,-ResX)[:,np.newaxis]
        GridX_B = np.append(GridX_B,LatLon_W_Meridian)

        GridY_B = np.ones(GridX_B.shape[0]-1).T*LPS_S_phi0
        GridY_B = np.append(GridY_B,LPS_S_phi0) #S

        # Compile values into a list
        X_vals = (GridX_T,GridX_B)
        Y_vals = (GridY_T,GridY_B)

    # If in the northern zone
    elif h == "N":
        # gridlines W->E
        GridX_T = np.arange(LatLon_W_Meridian,LatLon_E_Meridian,ResX)[:,np.newaxis]
        GridX_T = np.append(GridX_T,LatLon_E_Meridian)

        GridY_T = np.ones(GridX_T.shape[0]-1).T*LPS_N_Border
        GridY_T = np.append(GridY_T,LPS_N_Border)

        # draw E->W values
        GridX_B = np.arange(LatLon_E_Meridian,LatLon_W_Meridian,-ResX)[:,np.newaxis]
        GridX_B = np.append(GridX_B,LatLon_W_Meridian)

        GridY_B = np.ones(GridX_B.shape[0]-1).T*LPS_N_phi0
        GridY_B = np.append(GridY_B,LPS_N_phi0)#S

        # combine grid values into a list
        X_vals = (GridX_T,GridX_B)
        Y_vals = (GridY_T,GridY_B)

    # If no zone could be identified
    else:
        raise ValueError("LPS display Generation Terminated, Hemisphere could "
                         "not be identified.")

    # Concatenate grid values into a single list of values
    GridX = np.concatenate(X_vals)
    GridY = np.concatenate(Y_vals)

# generate grid that surrounds the LPS 80° border
# plotted in LPS Coordinates. This plots a circle in
# rectangular Coordinates
elif Grid_Res == "global":
    if h == "S":
        GridX = np.arange(LatLon_W_Meridian,LatLon_E_Meridian,ResX)[:,np.newaxis] #grid lines
        GridX = np.append(GridX,LatLon_E_Meridian)
```

```
            GridY = np.ones(GridX.shape[0]-1).T*LPS_S_Border
            GridY = np.append(GridY,LPS_S_Border)

        elif h == "N":
            GridX = np.arange(LatLon_W_Meridian,LatLon_E_Meridian,ResX)[:,np.newaxis] #grid lines
            GridX = np.append(GridX,LatLon_E_Meridian)

            GridY = np.ones(GridX.shape[0]-1).T*LPS_S_Border
            GridY = np.append(GridY,LPS_S_Border)

        else:
            raise ValueError("LPS Generation Terminated, Hemisphere and "
                             "grid type could not be identified.")
    # For program functionality and for exporting.
    # Output vectors are transformed from a 1D row vector
    # into a 2D array with a single row of values.
    GridX = GridX[np.newaxis,:]
    GridY = GridY[np.newaxis,:]

    # output LPS grids
    return GridX, GridY

# ----------------------------------------------------------------------
# SECTION 2.16: Generate LTM Grids values according to a specified
#               resolution and coordinate type


def generate_LTM_grid(Grid_Res,z=None,
                      h=None):
    """
    This function generates grids for the Lunar Transverse Mercator (LTM)
    system.

    Input Parameters:
    (int, float) Grid_Res - Grid format and resolution
    (int) z - LTM zone number designation
    (str) h - LTM hemispheric location designation

    Types of grids
    Grid_Res == global_all - Generate a multipolygon grid of all 90 LTM
                             zones plotted and dimensioned in LatLon
                             coordinate format.
                             Grid area is plotted as a box, and display
                             correctly the LatLon, LTM and LPS
                             projections.
    Grid_Res == global - Generate a single polygon of single LTM zone
                         plotted in LatLon in the associated LTM
                         projection. This border is drawn as a box and
                         can be displaced in LatLon, LPS and LTM
                         projections.
    Grid_Res == border - Generate a single polygon of single LTM zone
                         plotted in LTM Coordinates in the associated LTM
                         projection. This border is drawn as a box and
                         can be displaced in LatLon, LPS and LTM
                         projections but is designed for LTM only.
        Note: Global and Border grids are both written in latitude
```

```
        and longitude, the coordinate values are converted later
        in the function code to proper coordinate format.

Returns:
(2D array, floats) GridX -  Longitude or Easting values of the LTM
                    border.
(2D array, floats) GridY -  Latitude or Northing values of the LTM
                    border.
“””
# generate grid for all LTM zones in LatLon
if Grid_Res == “global_all”:
   k = 0 # unique id number for each polygon processed in
      # this function

   # Calculate dimensions of the array
   # Top and bottom, 8° width subdivided by ResX covered twice
   # Left and right 82° range subdivided by ResY per zone covered twice
   # Add for last appended ending values
   # each row is a polygon
   # each column is an LTM zone, 90 in total
   ArrayX = int(4 + (2*LTM_Width)/ResX + (2*82)/ResY) # dimension all points in
                                    # grid area
   ArrayY = 90 # dimension for total number of polygons

   # dimension NumPy arrays for storing and exporting grid values.
   GridX = np.zeros((ArrayY,ArrayX))
   GridY = np.zeros((ArrayY,ArrayX))

   # Loop through LTM grid areas based on LTM dimensions and generate
   # grid values for plotting

   # Iterate through the latitude range
   for i in range(LTM_S_Border,LTM_N_Border,LTM_N_Border):
      # iterate through the longitude range
      for j in range(LatLon_W_Meridian,LatLon_E_Meridian,LTM_Width):

         # determine LTM zone boundaries
         n = i + LTM_N_Border
         s = i
         e = j + LTM_Width
         w = j

         #write out grid values in W->E (Bottom)
         GridX_B = np.arange(w,e,ResX)[:,np.newaxis]
         GridX_B = np.append(GridX_B,w)

         GridY_B = np.ones(GridX_B.shape[0]-1).T*s
         GridY_B = np.append(GridY_B,s)

         #write out grid values in S -> N (right)
         GridY_R = np.arange(s,n,ResY)[:,np.newaxis]
         GridY_R = np.append(GridY_R,n)

         GridX_R = np.ones(GridY_R.shape[0]-1).T*e
         GridX_R = np.append(GridX_R,e)
```

```
            #write out grid values in E -> W (Top)
            GridX_T = np.arange(e,w,-ResX)[:,np.newaxis]
            GridX_T = np.append(GridX_T,w)

            GridY_T = np.ones(GridX_T.shape[0]-1).T*n
            GridY_T = np.append(GridY_T,n)

            #write out grid values in N -> S (left)
            GridY_L = np.arange(n,s,-ResY)[:,np.newaxis]
            GridY_L = np.append(GridY_L,s)

            GridX_L = np.ones(GridY_L.shape[0]-1).T*w
            GridX_L = np.append(GridX_L,w)

            # combine values into list
            X_vals = (GridX_B,GridX_R,GridX_T,GridX_L)
            Y_vals = (GridY_B,GridY_R,GridY_T,GridY_L)

            # concatenate values into single list
            # for exporting
            GridX[k,:] = np.concatenate(X_vals)
            GridY[k,:] = np.concatenate(Y_vals)

            k+=1 # iterate to the next polygon

# generate grid for a single LTM zone in LatLon,
# converted later to proper exporting format.
elif Grid_Res == "border" or Grid_Res == "global":
    # Calculate dimensions of the array:
    # Top and bottom, 8° width subdivided by ResX covered twice
    # Left and right 82° range subdivided by ResY per zone covered twice
    # Add for last appended ending values
    # each row is a polygon
    # each column is an LTM zone, 1 in total
    ArrayX = int(4 + (2*LTM_Width)/ResX + (2*82)/ResY)

    # Generate arrays based on calculated size
    GridX = np.zeros((ArrayX))
    GridY = np.zeros((ArrayX))

    # determine zone bounds in the north south range
    if h == "N": # if in the northern hemisphere of zones, phi >=0°
        n = LTM_N_Border
        s = LTM_Equator
    elif h  ==  "S": # if in the northern hemisphere of zones, phi <0°
        n = LTM_Equator - 1E-13   # integers calculated in adjacent zone.
                        # this is done correctly.
                        # To plot correctly we "force the value"
                        # into the correct zone but making is a
                        # very small value at matching precision
        s =  LTM_S_Border
    else:
        # if hemisphere could not be determined
        raise ValueError ("Correct LTM zone dimensions could not "
                    "be determined.")
```

```
    # calculate the east border value of the LTM zone
    e =  Calc_LTM_prime_meridian(z,"deg") + LTM_Width/2 - 1E-13
       # similarly, we subtract a small value to endure the
       # east zone is plotted correctly

    # calculate the west border value for the LTM zone
    w = Calc_LTM_prime_meridian(z,"deg") - LTM_Width/2

    # Write out grid X and Y values

    # bottom L -> R of the LTM zone
    GridX_B = np.arange(w,e,ResX)
    GridY_B = np.ones(GridX_B.shape[0])*s

    GridX_B = np.append(GridX_B,e)
    GridY_B = np.append(GridY_B,s)

    # right side of the LTM zone
    GridY_R = np.arange(s,n,ResY)
    GridX_R = np.ones(GridY_R.shape[0])*e

    GridX_R = np.append(GridX_R,e)
    GridY_R = np.append(GridY_R,n)

    # top edge of LTM zone
    GridX_T = np.arange(e,w,-ResX)
    GridY_T = np.ones(GridX_T.shape[0])*n

    GridX_T = np.append(GridX_T,w)
    GridY_T = np.append(GridY_T,n)

    #left side of the LTM zone
    GridY_L = np.arange(n,s,-ResY)
    GridX_L = np.ones(GridY_L.shape[0])*w

    GridY_L = np.append(GridY_L,s)
    GridX_L = np.append(GridX_L,w)

    # format into single row of vales
    X_vals = (GridX_B,GridX_R,GridX_T,GridX_L)
    Y_vals = (GridY_B,GridY_R,GridY_T,GridY_L)

    # concatenate values into a single list
    GridX[:] = np.concatenate(X_vals)
    GridY[:] = np.concatenate(Y_vals)

    # Convert Grid from 1D row vector into a 2D array
    # with a single row of values. Extra array axis
    # added to support exporting in this program
    GridX =  GridX[np.newaxis,:]
    GridY =  GridY[np.newaxis,:]

else:
    raise ValueError("LTM Generation Terminated, Grid type could "
                     "not be determined.")
# return LTM grid area
```

```
    return GridX, GridY

# ---------------------------------------------------------------------
# SECTION 2.17: Generate LGRS Grid values according to a specified
#               resolution and coordinate type


def generate_LGRS_grid(Grid_Res,lonBand=None,
                       h=None,
                       z1=None,
                       h1=None,
                       ulx=None,
                       uly=None,
                       z2=None,
                       h2=None,
                       lrx=None,
                       lry=None,
                       multigrid=False,
                       LatLon_override=False):
    """
       This function is used to generate associated grids for the Lunar Grid
    Reference System (LGRS) in all resolutions and formats.

    Input Parameters:
    (int, float) Grid_Res - Grid format and resolution to be generated
    (int) lonBand - LTM zone number designator for LGRS plot for grids
            25km or greater in resolution.
    (str) h - LTM hemispheric location designator for LGRS plot for grids
            25km or greater in resolution.
    (int) z1 - LTM zone number designator for upper left coordinate
             of grid dimensioning bounding box. Only used in grid less
             than or equal to 1km
    (str) h1 - LTM hemisphere designator for upper left coordinate
             of grid dimensioning bounding box. Only used in grid less
             than or equal to 1km
    (double) ulx - upper left x coordinate of the grid dimensioning
                bounding box. Only used in grids less than or equal
                to 1km.
    (double) uly - upper left y coordinate of the grid dimensioning
                bounding box. Only used in grids less than or equal
                to 1km.
    (int) z2 - LTM zone number designator for lower right coordinate
             of grid dimensioning bounding box. Only used in grid less
             than or equal to 1km
    (str) h2 - LTM hemisphere designator for lower right coordinate
             of grid dimensioning bounding box. Only used in grid less
             than or equal to 1km
    (double) lrx - Lower right x coordinate of the grid dimensioning
                bounding box. Only used in grids less than or equal
                to 1km.
    (double) lry - Lower right y coordinate of the grid dimensioning
                bounding box. Only used in grids less than or equal
                to 1km.
    (bool) multigrid - logical value used to create a grid that crosses
                    multiple zones in the LTM portion of LGRS only.
                    Once activated and set to True, the output format
```

```
                will be overwritten and forced to a LatLon
                coordinate structure in the defined projection for
                latitude. The coordinates will be forced to lat
                lon in this setting regardless if the grid is in
                only one zone. If False, z1, z2, h1, and h2 will
                be checked to ensure the grid is only in one zone.
                Program will terminate if the grid is in multi-
                pile zones.
(bool) LatLon_override - Logical value that will force the inverse
                    conversion of grid position back to lunar
                    latitude and longitude. the final exported
                    grid will be referenced to LatLon.
                    NOTE: the potential for incorrect projections,
                    the LGRS and LGRS grids are meant to be
                    utilized in the LPS and LTM PCRS. Without
                    utilization map grid distances will very and
                    map distortion will be uncalibrated. It is
                    recommended that this option not be utilized
                    to ensure correct positioning with the
                    designed LGRS PCRS (LPS and LTM).

Types of grids
   Grid_Res == global - Generate multipolygon grid of the global area
                grid of the LGRS plot within a single specified
                LTM zone. Typically this will generate 1, 8°x
                10°zone (labeled C or X) and 9, 8°x8° zones and
                is plotted in LTM Coordinates in a LatLon projection.
                multi grid not supported as it is not needed.
Grid_Res == global_all - Generate multipolygon grid of all global area
                grids in LGRS. This plots across all LTM zones in
                the latitude longitude coordinate format and in
                the associated projection. This setting results
                in the generation of 90 8°x10° LGRS global areas,
                and 810 8°x8° zones.
                multi grid not supported as it is not needed.
Grid_Res == 25km_all - Generate multipolygon grid of all 25km grid
                zones of LGRS. This plot is applied to the either
                LTM portion of LGRS and is calculated in the
                latitude longitude coordinate format with
                the associated projection. This setting results
                in the generation of 69920 25km grid zones.
                multigrid not supported as it is not needed.
Grid_Res == 25km - Generate multipolygon grid of all 25km grid
                zones in a single LTM zone of the LGRS. This
                grid is calculated in the LTM Coordinates in with
                the associated projection.
                multigrid not supported as it is not needed.
Grid_Res == 1km - Generate multipolygon 1km LGRS grid in
                a single or across multiple LTM zones. Grid is
                created in LTM Coordinates in with the
                associated projection. Upper left and lower right
                corner point required, multigrid is supported.
Grid_Res == 100m - Generate multipolygon 100m  LGRS grid in
                a single or across multiple LTM zones. Grid is
                created in LTM Coordinates in with the
                associated projection. Upper left and lower right
```

```
            corner point required, multigrid is supported.
Grid_Res == 10m - Generate multipolygon 10m  LGRS grid in
            a single or across multiple LTM zones. Grid is
            created in LTM Coordinates in with the
            associate projection. Upper left and lower right
            corner point required, multigrid is supported.
Grid_Res == 1m - Generate multipolygon 1m  LGRS grid in
            a single or across multiple LTM zones. Grid is
            created in LTM Coordinates in with the
            associated projection. Upper left and lower right
            corner point required, multigrid is supported.
   Note: This function is used to calculate the output grids for
       LGRS associated coordinate formats including LGRS,
       LGRS_ACC ( a full Artemis condensed format), and ACC
       with global location truncation.

Returns:
(2D array, floats) GridX - Longitude or Easting values of the LGRS
                    grid system contained in LTM coordinates
                    and contained within the correct LTM
                    border.
(2D array, floats) GridY - Latitude or Northing values of the LGRS
                    grid system contained in LTM coordinates
                    and contained within the correct LTM
                    border.
(bool) multigrid - if user specified processing a multigrid shape
                file return state to main function for later
                processing.
“””

# generate LGRS global area grid in LTM Coordinates
if Grid_Res == “global”: # LGRS 8° Areas in LTM Coordinates
                    # Coordinates
   # Calculate dimensions of the array
   # Top and bottom, 10° width subdivided by ResX covered twice (oversized)
   # Left and right 8° range subdivided by ResY per zone covered twice
   # Add for last appended ending values
   # each row is a polygon
   # each column is an LTM zone, 1 in total
   ArrayX = int(4+(2*LTM_Width)/ResX+(2*(LTM_Width+2))/ResY)
   ArrayY = int(10)

   # generate grids arrays to assign and export values from
   # NaN values are used to signify the grid polygon is complete
   # or that there is not a polygon in the row. This format is
   # used to support polygon clipping and merging
   GridX = np.full((ArrayY,ArrayX),np.nan)
   GridY = np.full((ArrayY,ArrayX),np.nan)

   # Determine LGRS  bounds based on LTM projection areas
   if h == “N”: # if in the northern portion
      N = LTM_N_Border-2 # provides a 0° to 80° range. Comparability for
                  # 10° zones
      S = LTM_Equator
   elif h == “S”: # if in the southern portion
      N = LTM_Equator
```

```
    S = LTM_S_Border+2 # provides a 0° to 80° range. Compatibility for
                # 10° zones
else:
    # Raise problem is hemisphere could not be determined
    raise ValueError (“Correct LTM zone dimensions for LGRS grid “
                “be determined.”)

k = 0 # unique identifier for polygon values
# iterate through the north south range of the LGRS global
# area grid, ie: 8° in latitude
for i in range(S,int(N),LTM_Width):

    # determine the central meridian of the LTM projection
    # area for processing
    lam0 = Calc_LTM_prime_meridian(lonBand,”deg”)

    # determine LGRS global area bounds
    n = i + LTM_Width # top
    s = i # bottom
    e = lam0 + LTM_Width/2 - 1E-13 # right, forced into zone by s
                    # subtracting small value
    w = lam0 - LTM_Width/2  # left

    # add in extended range, 8°x10°, LGRS global areas for
    # the north south range
    if s == -80:
        s =- 82
    elif n == 80:
        n = 82
    elif n == 0 and h == “S”:
        n -= 1E-13

    # generate arrays values for grid

    # write our L -> R to B
    GridX_B = np.arange(w,e,ResX)
    GridY_B = np.ones(GridX_B.shape[0])*s

    GridX_B = np.append(GridX_B,e)
    GridY_B = np.append(GridY_B,s)

    # write our B -> T to R
    GridY_R = np.arange(s,n,ResY)
    GridX_R = np.ones(GridY_R.shape[0])*e

    GridX_R = np.append(GridX_R,e)
    GridY_R = np.append(GridY_R,n)

    # write our R -> L to T
    GridX_T = np.arange(e,w,-ResX)
    GridY_T = np.ones(GridX_T.shape[0])*n

    GridX_T = np.append(GridX_T,w)
    GridY_T = np.append(GridY_T,n)

    # write our T -> B to L
```

```
        GridY_L = np.arange(n,s,-ResY)
        GridX_L = np.ones(GridY_L.shape[0])*w

        GridX_L = np.append(GridX_L,w)
        GridY_L = np.append(GridY_L,s)

        # combine all grid values into a list
        X_vals = (GridX_B,GridX_R,GridX_T,GridX_L)
        Y_vals = (GridY_B,GridY_R,GridY_T,GridY_L)

        # concatenate positioning values into a single list
        # this is a stack where each row is a single polygon
        X_vals = np.concatenate(X_vals)
        Y_vals = np.concatenate(Y_vals)

        # determine output array bounds
        X_vals_len = X_vals.shape[0]
        Y_vals_len = Y_vals.shape[0]

        # based on bounds assign value to polygon grid
        GridX[k,0:X_vals_len] = X_vals[0:X_vals_len]
        GridY[k,0:Y_vals_len] = Y_vals[0:Y_vals_len]

        k+=1 # iterate polygon identifier number

# 8°x8° and 8°x10° LGRS Areas in LatLon Coordinates
elif Grid_Res == “global_all”:

    # Calculate dimensions of the array
    # Top and bottom, 10° width subdivided by ResX covered twice (oversized)
    # Left and right 8° range subdivided by ResY per zone covered twice
    # Add for last appended ending values
    # each row is a polygon
    # All grid areas in Y. 360°/8°zones*20 grid areas per zone=900
    ArrayX = int(4 + (2*LTM_Width)/ResX + (2*(LTM_Width + 2))/ResY)
    ArrayY = 360*LTM_Width*20

    # generate grids arrays to assign and export values from
    # NaN values are used to signify the grid polygon is complete
    # or that there is not a polygon in the row. This format is
    # used to support polygon clipping and merging
    GridX = np.full((ArrayY,ArrayX),np.nan)
    GridY = np.full((ArrayY,ArrayX),np.nan)

    k = 0 # assign value to iterate through polygon values
    for i in range(int(LTM_S_Border) + 2,int(LTM_N_Border) - 2,LTM_Width):
        for j in range(int(LatLon_W_Meridian),int(LatLon_E_Meridian),LTM_Width):

            n = i + LTM_Width
            s = i
            e = j + LTM_Width - 1E-13
            w = j

            # add in extended range zones
            if s == -80:
                s =- 82
```

```
        elif n == 80:
            n = 82
        elif n == 0 and h == "S":
            n -= 1E-13

        # L -> R B
        GridX_B = np.arange(w,e,ResX)
        GridY_B = np.ones(GridX_B.shape[0])*s

        GridX_B = np.append(GridX_B,e)
        GridY_B = np.append(GridY_B,s)

        # B -> T R
        GridY_R = np.arange(s,n,ResY)
        GridX_R = np.ones(GridY_R.shape[0])*e

        GridX_R = np.append(GridX_R,e)
        GridY_R = np.append(GridY_R,n)

        # R -> L T
        GridX_T = np.arange(e,w,-ResX)
        GridY_T = np.ones(GridX_T.shape[0])*n

        GridX_T = np.append(GridX_T,w)
        GridY_T = np.append(GridY_T,n)

        # T -> B L
        GridY_L = np.arange(n,s,-ResY)
        GridX_L = np.ones(GridY_L.shape[0])*w

        GridX_L = np.append(GridX_L,w)
        GridY_L = np.append(GridY_L,s)

        # combine parts of the polygon into a list
        X_vals = (GridX_B,GridX_R,GridX_T,GridX_L)
        Y_vals = (GridY_B,GridY_R,GridY_T,GridY_L)

        # concatenate polygons together into a single
        # row
        X_vals = np.concatenate(X_vals)
        Y_vals = np.concatenate(Y_vals)

        # determine number of grid values for assignment
        X_vals_len = X_vals.shape[0]
        Y_vals_len = Y_vals.shape[0]

        # assign grid values to array of polygons
        GridX[k,0:X_vals_len] = X_vals[0:X_vals_len]
        GridY[k,0:Y_vals_len] = Y_vals[0:Y_vals_len]

        k+=1 # iterate to the next polygon

# grid of 25km grid zone in all LTM zones in LatLon
elif Grid_Res == "25km_all":

    # Calculate dimensions of the array
```

```
# Top and bottom, 10° width subdivided by ResX covered twice (oversized)
# Left and right 8° range subdivided by ResY per zone covered twice
# Add for last appended ending values
# each row is a polygon
# All grid areas in Y. 360°/8°zones*20 grid areas per zone=900
      #   + 100 for extra global areas
# 90 LTM zones as well

ArrayX = int(10/ResX/2+100) # 10 degrees divided by resolution.
                # a safe number to represent the curvature of an
                # LGRS LTM zone.

ArrayY = int(FalseEasting/25E3 * FalseNorthing/25E3 - 240 + 100)
              # No 25km grid squares in LTM zone
              # the max is 1000 per LTM zone
              # but most are clipped and 760
              # should be the max

ArrayZ = int(360/LTM_Width*2)  # 90 LTM zones

# dimension arrays,( LTM zones
#                x 25km grids zones per LTM zone
#                x estimated max polygon length )
# NaNs are used to close the polygon or used to show
# now polygon was assigned
GridX = np.full((ArrayZ,ArrayY,ArrayX),np.nan)
GridY = np.full((ArrayZ,ArrayY,ArrayX),np.nan)

k = 0 # for iterating through unique polygons

# Loop through all LTM zones
for i in range(int(LTM_S_Border),int(LTM_N_Border),int(LTM_N_Border)):
   for j in range(int(LatLon_W_Meridian),int(LatLon_E_Meridian),LTM_Width):

      z = Calc_LTM_zone(j + LTM_Width/2)
      lam0 = Calc_LTM_prime_meridian(z,"deg")

      n = i + LTM_N_Border
      s = i
      e = j + LTM_Width - 1E-13
      w = j

      if s < 0:
         h = 'S'
      elif s >= 0:
         h = 'N'

      if n == 0 and h == "S":
         n -= 1E-13

      # dimensions used for dimensioning data array
      X_max = FalseEasting
      X_min = 0
      Y_max = FalseNorthing
      Y_min = 0
```

```
            # generate arrays
            Grid_X = np.arange(X_min,X_max,25E3)
            Grid_Y = np.arange(Y_min,Y_max,25E3)

            Grid_X = np.append(Grid_X,X_max)
            Grid_Y = np.append(Grid_Y,Y_max)

            # format meshgrid for all zones
            Cylindrical_X,Cylindrical_Y = np.meshgrid(Grid_X,Grid_Y)

            # shift X coordinates so grid is centered in LTM zone
            Cylindrical_X+=FalseEasting/2

            # assign grid to geometry
            LGRS_25km_geom_single_area = assign_xy_grid_to_multipolygon(Cylindrical_X,Cylindrical_Y)

            # collect global areas and merge
            GlobalX,GlobalY,multigrid = generate_LGRS_grid("global",z,h)
            # convert global zone boundary coordinates from LatLon to LTM
            for m in range(0,GlobalX.shape[0]):
               for n in range(0,GlobalY.shape[1]):
                  if np.isnan(GlobalX[m,n]):
                     continue
                  z,h,GlobalX[m,n],GlobalY[m,n] = toLTM(GlobalX[m,n],GlobalY[m,n],zone=None,tr
unc_val=0,process_errors=False)
            # assign to mutlipolygon
            LGRS_Global_geom = assign_arbitrary_xy_to_multipolygon(GlobalX,GlobalY)
            # union areas into
            LGRS_25kmGlobal_geom_single_area = intersect_multipolygons(LGRS_25km_geom_single_area,LGRS_
Global_geom)

            # determine LTM bounding box for clipping
            LTM_X,LTM_Y = generate_LTM_grid("border",z,h)

            # convert coordinates to LTM values
            for m in range(0,LTM_X.shape[0]):
               for n in range(0,LTM_X.shape[1]):
                  z,h,LTM_X[m,n],LTM_Y[m,n] = toLTM(LTM_X[m,n],LTM_Y[m,n],zone=None,trunc_val=0,process_erro
rs=False)
            LTM_zone_clip_geom = assign_xy_to_polygon(LTM_X,LTM_Y)

            # clip to bounding box
            LGRS_area_clipped = clip_multipolygon_by_polygon(LGRS_25kmGlobal_geom_single_area,LTM_zone_clip_
geom)

            # extract X, Y positions from clipped region
            X_vals,Y_vals = extract_xy_from_multipolygon(LGRS_area_clipped)

            GridX[k,0:X_vals.shape[0],0:X_vals.shape[1]] = X_vals
            GridY[k,0:X_vals.shape[0],0:X_vals.shape[1]] = Y_vals

            # convert coordinates to LatLon values for exporting
            for mi in range(0,GridX.shape[1]):
               for ni in range(0,GridX.shape[2]):
                  GridX[k,mi,ni],GridY[k,mi,ni] = toLatLon(GridX[k,mi,ni],GridY[k,mi,ni],z,h,process_errors=True)
```

```
            k+=1

        # Reshape the 3D array to a 2D array
        GridX = np.vstack(GridX) #[0,:,:] #.reshape(GridX.shape[0], -1)
        GridY = np.vstack(GridY) #[0,:,:] #.reshape(GridY.shape[0], -1)

    # calculate all 25km grid areas in a single LGRS LTM zone in LTM
    # coordinates
    elif Grid_Res == "25km":
        if lonBand == None or h == None:
            raise ValueError ("LGRS 25km single global zone is projected "
                              "A LTM zone is required for plotting "
                              "operation: Z\{1-45\} h\{N|S\}")

        # Calculate dimensions of the array
        # Top and bottom, 10° width subdivided by ResX covered twice (oversized)
        # Left and right 8° range subdivided by ResY per zone covered twice
        # Add for last appended ending values
        # each row is a polygon
        # All grid areas in Y. 360°/8°zones*20 grid areas per zone=900
        # 90 LTM zones as well
        ArrayX = int(10/ResX/2 +18010) # 10 degrees divided by resolution.
                         # a safe number to represent the curvature of an
                         # LGRS LTM zone.

        ArrayY = int(FalseEasting/25E3*FalseNorthing/25E3 - 240 + 1000)
                     # No 25km grid squares in LTM zone
                     # the max is 1000 per LTM zone
                     # but most are clipped and 760
                     # should be the max

        # dimension grid values
        GridX = np.full((ArrayY,ArrayX),np.nan)
        GridY = np.full((ArrayY,ArrayX),np.nan)

        # determine LTM zone boundaries based on hemisphere
        if h == "N":
            n = LTM_N_Border
            s = LTM_Equator
        elif h == "S":
            n = LTM_Equator - 1E-13   # integers calculated in adjacent zone.
                              # this is done correctly.
                              # To plot correctly we "force the value"
                              # into the correct zone
            s = LTM_S_Border
        else:
            raise ValueError ("Correct LTM zone dimensions could not "
                              "be determined.")

        # determine east west range base on the LGRS lonBand
        e = Calc_LTM_prime_meridian(lonBand,"deg")+LTM_Width/2-1E-13
        w = Calc_LTM_prime_meridian(lonBand,"deg")-LTM_Width/2

        # Generate 25km grid values
        X_max = FalseEasting
        X_min = 0
```

```
            Y_max = FalseNorthing
            Y_min = 0

            # generate arrays
            Grid_X = np.arange(X_min,X_max,25E3)
            Grid_Y = np.arange(Y_min,Y_max,25E3)

            Grid_X = np.append(Grid_X,X_max)
            Grid_Y = np.append(Grid_Y,Y_max)

            # format mesh grid for all zones
            Cylindrical_X,Cylindrical_Y = np.meshgrid(Grid_X,Grid_Y)
            Cylindrical_X += FalseEasting/2

            # assign grid to geometry
            LGRS_25km_geom_single_area = assign_xy_grid_to_multipolygon(Cylindrical_X,Cylindrical_Y)

            # collect global areas and merge
            GlobalX,GlobalY,multigrid = generate_LGRS_grid("global",lonBand,h)
            # convert global zone boundary coordinates from LatLon to LTM
            for m in range(0,GlobalX.shape[0]):
               for n in range(0,GlobalY.shape[1]):
                  if np.isnan(GlobalX[m,n]):
                     continue
                  z,h,GlobalX[m,n],GlobalY[m,n] = toLTM(GlobalX[m,n],GlobalY[m,n],zone=None,tr
unc_val=0,process_errors=False)
            # assign to mutlipolygon
            LGRS_Global_geom = assign_arbitrary_xy_to_multipolygon(GlobalX,GlobalY)
            # union areas into
            LGRS_25kmGlobal_geom_single_area = intersect_multipolygons(LGRS_25km_geom_single_area,LGRS_Global_
geom)

            # determine LTM bounding box for clipping
            LTM_X,LTM_Y = generate_LTM_grid("border",lonBand,h)
            # convert LTM zone boundary coordinates from LatLon to LTM
            for m in range(0,LTM_X.shape[0]):
               for n in range(0,LTM_X.shape[1]):
                  z,h,LTM_X[m,n],LTM_Y[m,n] = toLTM(LTM_X[m,n],LTM_Y[m,n],zone=None,tr
unc_val=0,process_errors=False)

            # clip 25km grid to LTM zone boundaries
            LTM_zone_clip_geom = assign_xy_to_polygon(LTM_X,LTM_Y)

            # clip to bounding box
            LGRS_area_clipped = clip_multipolygon_by_polygon(LGRS_25kmGlobal_geom_single_area,LTM_zone_clip_geom)

            # extract X, Y positions from clipped region
            X_vals,Y_vals = extract_xy_from_multipolygon(LGRS_area_clipped)

            # Assign final values to grid in LTM coordinates
            GridX[0:X_vals.shape[0],0:X_vals.shape[1]] = X_vals
            GridY[0:Y_vals.shape[0],0:Y_vals.shape[1]] = Y_vals

         # plotting smaller grids less than 25km resolution
         elif Grid_Res == "1km" or \
              Grid_Res == "100m" or \
```

```
 Grid_Res == "10m" or \
 Grid_Res == "1m":

# determine grid size in meters not a string
if Grid_Res == "1km":
   grid_sizeY,grid_sizeX = 1E3,1E3
elif Grid_Res == "100m":
   grid_sizeY,grid_sizeX = 100,100
elif Grid_Res == "10m":
   grid_sizeY,grid_sizeX = 10,10
elif Grid_Res == "1m":
   grid_sizeY,grid_sizeX = 1,1

if ulx == None or uly == None or lrx == None or lry == None:
   raise ValueError ("LGRS 1km grid requires an input box "
                "provide ulx,uly,lrx,lry")

# generate single zone grid: extend to clipping border.
# for simplicity the grids are contained within a whole zone
merge_grid = None
# if merge grid is activated than the geometries of multiple
# polygon geometries will be merged together in a union of
# spaces.

# If the multigrid setting is not enabled check to make
# sure
if not multigrid:
   if z1 != z2 or h1 != h2:
      choice = input("Map area crosses multiple zones "
            "would you like export Coordinates in LatLon to "
            "support multiple Zones? (y/n): ")

      # Double check that a multigrid plot is not wanted.
      # Terminate if not
      if choice.lower() != 'y':
         raise ValueError ("Plot Area crosses multiple zones: enable multigrid "
                     "setting to plot this area")
      else:
         multigrid = True

# check if there is a wrap around the globe in a multigrid plot
if z1>z2:
   # string used to iterate through zones
   zone_range = list(range(z1,45+1,1))+list(range(1,z2+1,1))
   rev = True
else:
   # normal zone range for multi plot. This works in one
   # zone if z1 and z2 are the same
   zone_range = range(z1,z2+1,1)
   rev = False

# iterate inducing and between z1 and z2
for z in zone_range:

   # double check if the plot area crosses multiple hemispheres
   # and generate iteration list
```

```
if h1 == h2:
   hems = [h1]
else:
   hems = [h1,h2]

# iterate through grid box hemispheres
for h in hems:
   LGRS_grid_clipped = None
   lam0 = Calc_LTM_prime_meridian(z,”deg”)
   # create box for new Coordinates to cross zones

   # correct X position for multiple zones across the -180-180 meridian
   if rev and z == 1:
      ztemp = z1
      z2 = ztemp
      z2 = z1

   # create left most corner of grid area
   if z == z1:
      Cylindrical_Xmin = grid_sizeX*(ulx//grid_sizeX)
   else: # create left most area to confine to LTM zone
      _,_,Cylindrical_Xmin,_ = toLTM(lam0-4,0,zone=None,trunc_val=0,process_errors=False)
      Cylindrical_Xmin = grid_sizeX*(Cylindrical_Xmin//grid_sizeX)

   # create right portion of grid area
   if z == z2:
      Cylindrical_Xmax = grid_sizeX*(lrx//grid_sizeX+1)
   else: # create right most area to confine to LTM zone
      _,_,Cylindrical_Xmax,_ = toLTM(lam0+4-1E-13,0,zone=None,trunc_val=0,process_errors=False)
      Cylindrical_Xmax = grid_sizeX*(Cylindrical_Xmax//grid_sizeX+1)

   # determine min and max of the grid area and if needed
   # process hemispheres for grid generation in both
   # hemispheres
   if h1 == h2:
      Cylindrical_Ymin = grid_sizeY*(lry//grid_sizeY)
      Cylindrical_Ymax = grid_sizeY*(uly//grid_sizeY+1)

   elif h == “S”: # format bottom of the grid
      Cylindrical_Ymin = grid_sizeY*(lry//grid_sizeY+1)

      Cylindrical_Ymax = FalseNorthing
      Cylindrical_Ymax = grid_sizeY*(Cylindrical_Ymax//grid_sizeY)

   elif h == “N”: # format top of the grid
      Cylindrical_Ymax = grid_sizeY*(uly//grid_sizeY+1)

      Cylindrical_Ymin = 0
      Cylindrical_Ymin = grid_sizeY*(Cylindrical_Ymin//grid_sizeY)

   # Generate grid
   GridY = np.arange(Cylindrical_Ymin,Cylindrical_Ymax,grid_sizeY)
   GridX = np.arange(Cylindrical_Xmin,Cylindrical_Xmax,grid_sizeX)

   # add in final endpoint to ensure correct positioning
   GridY = np.append(GridY,Cylindrical_Ymax)
```

```
        GridX = np.append(GridX,Cylindrical_Xmax)

        # create grid
        GridX,GridY = np.meshgrid(GridX,GridY)

        # convert LTM Coordinates to LatLon values
        # this supports a multigrid plot. It is reconverted later
        # if contained in a single zone
        for i in range(0,GridX.shape[0]):
            for j in range(0,GridX.shape[1]):
                GridX[i,j],GridY[i,j] = toLatLon(GridX[i,j],GridY[i,j],
                                        zone=z,
                                        h=h,
                                        process_errors=False)

        # assign values to polygon
        LGRS_grid_area = assign_xy_grid_to_multipolygon(GridX,GridY)
        # collect global areas and merge

        GlobalX,GlobalY,_ = generate_LGRS_grid("global",z,h)

        # assign to mutlipolygon
        LGRS_Global_geom = assign_arbitrary_xy_to_multipolygon(GlobalX,GlobalY)
        # # union areas into
        LGRS_AreaGlobal_geom_single_area = intersect_multipolygons(LGRS_grid_area,LGRS_Global_geom)

        # determine LTM bounding box for clipping
        LTM_X,LTM_Y = generate_LTM_grid("border",z,h)
        # print(LTM_X,LTM_Y)

        # assign to geometry for clipping
        LTM_zone_clip_geom = assign_xy_to_polygon(LTM_X,LTM_Y)

        # clip to bounding box geometry
        LGRS_grid_clipped = clip_multipolygon_by_polygon(LGRS_AreaGlobal_geom_single_area,LTM_zone_clip_
geom)

        # processing flag to not call the merging function
        # on a multipolygon in the first grid area
        if merge_grid == None:
            merge_grid = LGRS_grid_clipped
        else:
            # merge multiple polygons together.
            # output is the input on the next loop iteration.
            merge_grid = merge_multipolygons(merge_grid,LGRS_grid_clipped)

    # print(LGRS_grid_area)

    # extract X, Y positions from clipped region
    GridX,GridY = extract_xy_from_multipolygon(merge_grid)

    # convert back to LTM Coordinates. if plot crosses multiple zones
    # leave in lat and lon
    if not multigrid:
        for i in range(0,GridX.shape[0]):
            for j in range(0,GridX.shape[1]):
```

```
                if np.isnan(GridX[i,j]):
                    continue
                _,_,GridX[i,j],GridY[i,j] = toLTM(GridX[i,j],GridY[i,j],zone=None,
                                    trunc_val=0,process_errors=False)

    else:
        raise ValueError("LGRS Generation Terminated: Grid type could"
                         "not be determined. Please specify:\n"
                         "\{global,global_all,25km,25km_all,1km,100m, "
                         "10m, or 1m \}")

    # functionality to force the output grid into lunar LatLon
    # (not recommended as the grid will not be tied to a PCRS)
    if LatLon_override:
        if (not multigrid) and \
          ((Grid_Res != "global_all") and \
          (Grid_Res != "25km_all")  and \
          (Grid_Res != "global")):

            for i in range(0,GridX.shape[0]):
                for j in range(0,GridY.shape[1]):
                    if np.isnan(GridX[i,j]):
                        continue
                    GridX[i,j],GridY[i,j]=toLatLon(
                                        GridX[i,j],
                                        GridY[i,j],
                                        z1,
                                        h,
                                        process_errors=True)
        else:
            raise ValueError("LatLon Override could not be applied as "
                             "grids are already in a LatLon format:\n"
                             "Set latlon_override to False to continue")

    return GridX, GridY, multigrid

# ----------------------------------------------------------------------
# SECTION 2.18: Generate LGRS polar grid values according to a specified
#               resolution and coordinate type


def generate_PolarLGRS_grid(Grid_Res,h=None,
                            h1=None,
                            ulx=None,
                            uly=None,
                            h2=None,
                            lrx=None,
                            lry=None,
                            LatLon_override=False):
    """
     This function is used to generate associated grids for the Lunar Grid
    Reference System (LGRS) in all resolutions and formats for the LPS
    portions of the grid system.

    Input Parameters:
    (int, float) Grid_Res - Grid format and resolution to be generated
```

(str) h - LPS hemispheric location designator for LGRS plot for grids
25km or greater in resolution.
(str) h1 - LPS hemisphere designator for upper left coordinate
of grid dimensioning bounding box. Only used in grid less
than or equal to 1km
(double) ulx - upper left x coordinate of the grid dimensioning
bounding box. Only used in grids less than or equal
to 1km.
(double) uly - upper left y coordinate of the grid dimensioning
bounding box. Only used in grids less than or equal
to 1km.
(str) h2 - LPS hemisphere designator for lower right coordinate
of grid dimensioning bounding box. Only used in grid less
than or equal to 1km
(double) lrx - Lower right x coordinate of the grid dimensioning
bounding box. Only used in grids less than or equal
to 1km.
(double) lry - Lower right y coordinate of the grid dimensioning
bounding box. Only used in grids less than or equal
to 1km.
(bool) LatLon_override - Logical value that will force the inverse
conversion of grid position back to lunar
latitude and longitude. the final exported
grid will be referenced to LatLon.
NOTE: the potential for incorrect projections,
the LGRS and LGRS grids are meant to be
utilized in the LPS and LTM PCRS. Without
utilization map grid distances will very and
map distortion will be uncalibrated. It is
recommended that this option not be utilized
to ensure correct positioning with the
designed LGRS PCRS (LPS and LTM).

Types of grids
Grid_Res == global - Generate multipolygon grid of the global area
grid of the LGRS plot within a single specified
LPS zone. This is a two-part plot of the global
easting value of the north pole and is plotted
in LPS Coordinates in a stereographic projection.
Grid_Res == global_all - Generate multipolygon grid of all global area
grids in LGRS. This plots across a single specified
LPS zone. This is a two-part plot of the global
easting value of the north pole and is plotted
in LatLon Coordinates in a stereographic projection.
Grid_Res == 25km - Generate multipolygon grid of all 25km grid
areas of PolarLGRS in a single LPS zone.
This plot is calculated in the LPS coordinate format
with with the associated projection
Grid_Res == 1km - Generate multipolygon 1km PolarLGRS grid in
a single LPS zones. Grid is created in LPS
Coordinates in with the associated projection.
Upper left and lower right corner point required.
Grid_Res == 100m - Generate multipolygon 100m LGRS grid in
a single LPS zones. Grid is created in LPS
Coordinates in with the associated projection.
Upper left and lower right corner point required.

```
Grid_Res == 10m - Generate multipolygon 10m LGRS grid in
            a single LPS zones. Grid is created in LPS
            Coordinates in with the associated projection.
            Upper left and lower right corner point required.
Grid_Res == 1m - Generate multipolygon 1m LGRS grid in
            a single LPS zones. Grid is created in LPS
            Coordinates in with the associated projection.
            Upper left and lower right corner point required.
   Note: This function is used to calculate the output grids for
        LGRS associated coordinate formats inducing PolarLGRS,
        PolarLGRS_ACC (a full Artemis condensed format), and
        PolarACC with global location truncation.

Returns:
(2D array, floats) GridX - Longitude or Easting values of the
                    polar LGRS grid in LPS coordinate. Grid
                    clipped within the LPS borders.
(2D array, floats) GridY - Latitude or Northing values of the
                    polar LGRS grid in LPS coordinate. Grid
                    clipped within the LPS borders.
“””
# LGRS polar zones, in a single LPS zone LPS coordinates
if Grid_Res == “global”:

   if h ==  “S”:
      # dimensions grid size by maximum possible size of the grid in X
      # for only two polygons
      GridX = np.full((2,int(2*(LPS_S_Border-LPS_S_phi0)/ResY+
                  int(LatLon_E_Meridian-LatLon_W_Meridian/ResX
                  +3))),np.nan)
      # dimensions grid size by maximum possible size of the grid in Y
      # for only two polygons
      GridY = np.full((2,int(2*(LPS_S_Border-LPS_S_phi0)/ResY+
                  int(LatLon_E_Meridian-LatLon_W_Meridian/ResX+
                  3))),np.nan)

      # Polar LGRS Poly 1
      # Northern border
      GridX_B = np.arange(LatLon_W_Meridian,0,ResX)[:,np.newaxis] #grid lines
      GridX_B = np.append(GridX_B,0)

      GridY_B = np.ones(GridX_B.shape[0]-1).T*LPS_S_Border
      GridY_B = np.append(GridY_B,LPS_S_Border)

      # Crossing the hemisphere
      GridY_C1 = np.arange(LPS_S_Border,LPS_S_phi0,-ResY)[:,np.newaxis] #grid lines
      GridY_C1 = np.append(GridY_C1,LPS_S_phi0)

      GridX_C1 = np.ones(GridY_C1.shape[0]-1).T*0
      GridX_C1 = np.append(GridX_C1,0)


      GridY_C2 = np.arange(LPS_S_phi0,LPS_S_Border,ResY)[:,np.newaxis] #grid lines
      GridY_C2 = np.append(GridY_C2,LPS_S_Border)

      GridX_C2 = np.ones(GridY_C2.shape[0]-1).T*LatLon_W_Meridian
```

```
    GridX_C2 = np.append(GridX_C2,LatLon_W_Meridian)

    # combine grid values and concatenate into single list
    X_vals = (GridX_B,GridX_C1,GridX_C2)
    X_vals = np.concatenate(X_vals)

    Y_vals = (GridY_B,GridY_C1,GridY_C2)
    Y_vals = np.concatenate(Y_vals)

    # assign values to output arrays
    GridX[0,0:X_vals.shape[0]] = X_vals[:]
    GridY[0,0:Y_vals.shape[0]] = Y_vals[:]

    # Polar LGRS Poly 2
    GridX_B = np.arange(0,LatLon_E_Meridian,ResX)[:,np.newaxis] #grid lines
    GridX_B = np.append(GridX_B,LatLon_E_Meridian)

    GridY_B = np.ones(GridX_B.shape[0]-1).T*LPS_S_Border
    GridY_B = np.append(GridY_B,LPS_S_Border)

    # Crossing the hemisphere
    GridY_C1 = np.arange(LPS_S_phi0,LPS_S_Border,ResY)[:,np.newaxis] #grid lines
    GridY_C1 = np.append(GridY_C1,LPS_S_phi0)

    GridX_C1 = np.ones(GridY_C1.shape[0]-1).T*0
    GridX_C1 = np.append(GridX_C1,0)

    GridY_C2 = np.arange(LPS_S_Border,LPS_S_phi0,-ResY)[:,np.newaxis] #grid lines
    GridY_C2 = np.append(GridY_C2,LPS_S_phi0)

    GridX_C2 = np.ones(GridY_C2.shape[0]-1).T*LatLon_W_Meridian
    GridX_C2 = np.append(GridX_C2,0)

    # combine grid values and concatenate into single list
    X_vals = (GridX_B,GridX_C1,GridX_C2)
    X_vals = np.concatenate(X_vals)

    Y_vals = (GridY_B,GridY_C1,GridY_C2)
    Y_vals = np.concatenate(Y_vals)

    # assign values to output arrays
    GridX[1,0:X_vals.shape[0]] = X_vals[:]
    GridY[1,0:Y_vals.shape[0]] = Y_vals[:]

elif h == “N”:
    # dimensions grid size by maximum possible size of the grid in X
    # for only two polygons
    GridX = np.full((2,int(2*(LPS_N_phi0-LPS_N_Border)/ResY+
                int(LatLon_E_Meridian-LatLon_W_Meridian/ResX
                +3))),np.nan)
    # dimensions grid size by maximum possible size of the grid in Y
    # for only two polygons
    GridY = np.full((2,int(2*(LPS_N_phi0-LPS_N_Border)/ResY+
                int(LatLon_E_Meridian-LatLon_W_Meridian/ResX+
                3))),np.nan)
```

```
# Poly 1
# Northern border
GridX_B = np.arange(LatLon_W_Meridian,0,ResX)[:,np.newaxis] #grid lines
GridX_B = np.append(GridX_B,0)

GridY_B = np.ones(GridX_B.shape[0]-1).T*LPS_N_Border
GridY_B = np.append(GridY_B,LPS_N_Border)

# Crossing the hemisphere

GridY_C1 = np.arange(LPS_N_phi0,LPS_N_Border,-ResY)[:,np.newaxis] #grid lines
GridY_C1 = np.append(GridY_C1,LPS_N_Border)

GridX_C1 = np.ones(GridY_C1.shape[0]-1).T*0
GridX_C1 = np.append(GridX_C1,0)


GridY_C2 = np.arange(LPS_N_Border,LPS_N_phi0,ResY)[:,np.newaxis] #grid lines
GridY_C2 = np.append(GridY_C2,LPS_N_phi0)

GridX_C2 = np.ones(GridY_C2.shape[0]-1).T*LatLon_W_Meridian
GridX_C2 = np.append(GridX_C2,LatLon_W_Meridian)

# combine grid values and concatenate into single list
X_vals = (GridX_B,GridX_C1,GridX_C2)
X_vals = np.concatenate(X_vals)
Y_vals = (GridY_B,GridY_C1,GridY_C2)
Y_vals = np.concatenate(Y_vals)

# assign values to output arrays
GridX[0,0:X_vals.shape[0]] = X_vals[:]
GridY[0,0:Y_vals.shape[0]] = Y_vals[:]

# grid zone 2
GridX_B = np.arange(0,LatLon_E_Meridian,ResX)[:,np.newaxis] #grid lines
GridX_B = np.append(GridX_B,LatLon_E_Meridian)

GridY_B = np.ones(GridX_B.shape[0]-1).T*LPS_N_Border
GridY_B = np.append(GridY_B,LPS_N_Border)

# Crossing the hemisphere
GridY_C1 = np.arange(LPS_N_Border,LPS_N_phi0,ResY)[:,np.newaxis] #grid lines
GridY_C1 = np.append(GridY_C1,LPS_N_Border)

GridX_C1 = np.ones(GridY_C1.shape[0]-1).T*0
GridX_C1 = np.append(GridX_C1,0)


GridY_C2 = np.arange(LPS_N_phi0,LPS_N_Border,-ResY)[:,np.newaxis] #grid lines
GridY_C2 = np.append(GridY_C2,LPS_N_Border)

GridX_C2 = np.ones(GridY_C2.shape[0]-1).T*LatLon_W_Meridian
GridX_C2 = np.append(GridX_C2,0)

# combine grid values and concatenate into single list
X_vals = (GridX_B,GridX_C1,GridX_C2)
```

```
        X_vals = np.concatenate(X_vals)
        Y_vals = (GridY_B,GridY_C1,GridY_C2)
        Y_vals = np.concatenate(Y_vals)

        # assign values to output arrays
        GridX[1,0:X_vals.shape[0]] = X_vals[:]
        GridY[1,0:Y_vals.shape[0]] = Y_vals[:]

    # end processing if grid area could not be identified
    else:
        raise ValueError("LPS Global plot could not be plotted.")

# LGRS polar zones in single LPS zone LatLon Coordinates
elif Grid_Res == "global_all":
    if h == "S":
        # dimensions grid size by maximum possible size of the grid in X
        # for only two polygons
        GridX = np.full((2,int(2*(LPS_S_Border-LPS_S_phi0)/ResY+
                    int(2*(LatLon_E_Meridian-LatLon_W_Meridian)/ResX
                    +4))),np.nan)
        # dimensions grid size by maximum possible size of the grid in Y
        # for only two polygons
        GridY = np.full((2,int(2*(LPS_S_Border-LPS_S_phi0)/ResY+
                    int(2*(LatLon_E_Meridian-LatLon_W_Meridian)/ResX+
                    4))),np.nan)

        # Poly 1
        # L - > R border
        GridX_B = np.arange(LatLon_W_Meridian,0,ResX)[:,np.newaxis] #grid lines
        GridX_B = np.append(GridX_B,0)

        GridY_B = np.ones(GridX_B.shape[0]-1).T*LPS_S_phi0
        GridY_B = np.append(GridY_B,LPS_S_phi0)

        # S -> N
        GridY_R = np.arange(LPS_S_phi0,LPS_S_Border,ResY)[:,np.newaxis] #grid lines
        GridY_R = np.append(GridY_R,LPS_S_Border)

        GridX_R = np.ones(GridY_R.shape[0]-1).T*0
        GridX_R = np.append(GridX_R,0)

        # R -> L
        GridX_T = np.arange(0,LatLon_W_Meridian,-ResX)[:,np.newaxis] #grid lines
        GridX_T = np.append(GridX_T,LatLon_W_Meridian)

        GridY_T = np.ones(GridX_T.shape[0]-1).T*LPS_S_Border
        GridY_T = np.append(GridY_T,LPS_S_Border)

        # N -> S
        GridY_L = np.arange(LPS_S_Border,LPS_S_phi0,-ResY)[:,np.newaxis] #grid lines
        GridY_L = np.append(GridY_L,LPS_S_phi0)

        GridX_L = np.ones(GridY_L.shape[0]-1).T*LatLon_W_Meridian
        GridX_L = np.append(GridX_L,LatLon_W_Meridian)

        # combine parts of the polygon into list
```

```
    X_vals = (GridX_B,GridX_R,GridX_T,GridX_L)
    Y_vals = (GridY_B,GridY_R,GridY_T,GridY_L)

    # combine polygon into a single list
    X_vals = np.concatenate(X_vals)
    Y_vals = np.concatenate(Y_vals)

    # assigning grid to output arrays
    GridX[0,0:X_vals.shape[0]] = X_vals[:]
    GridY[0,0:Y_vals.shape[0]] = Y_vals[:]


    # Poly 2
    GridX_B = np.arange(0,LatLon_E_Meridian,ResX)[:,np.newaxis] #grid lines
    GridX_B = np.append(GridX_B,LatLon_E_Meridian)

    GridY_B = np.ones(GridX_B.shape[0]-1).T*LPS_S_phi0
    GridY_B = np.append(GridY_B,LPS_S_phi0)

    # S -> N
    GridY_R = np.arange(LPS_S_phi0,LPS_S_Border,ResY)[:,np.newaxis] #grid lines
    GridY_R = np.append(GridY_R,LPS_S_Border)

    GridX_R = np.ones(GridY_R.shape[0]-1).T*LatLon_E_Meridian
    GridX_R = np.append(GridX_R,LatLon_E_Meridian)

    # R -> L
    GridX_T = np.arange(LatLon_E_Meridian,0,-ResX)[:,np.newaxis] #grid lines
    GridX_T = np.append(GridX_T,LatLon_W_Meridian)

    GridY_T = np.ones(GridX_T.shape[0]-1).T*LPS_S_Border
    GridY_T = np.append(GridY_T,LPS_S_Border)

    # N -> S
    GridY_L = np.arange(LPS_S_Border,LPS_S_phi0,-ResY)[:,np.newaxis] #grid lines
    GridY_L = np.append(GridY_L,LPS_S_phi0)

    GridX_L = np.ones(GridY_L.shape[0]-1).T*0
    GridX_L = np.append(GridX_L,0)
    # combine parts of the polygon into list
    X_vals = (GridX_B,GridX_R,GridX_T,GridX_L)
    Y_vals = (GridY_B,GridY_R,GridY_T,GridY_L)

    # combine polygon into a single list
    X_vals = np.concatenate(X_vals)
    Y_vals = np.concatenate(Y_vals)

    # assigning grid to output arrays
    GridX[1,0:X_vals.shape[0]] = X_vals[:]
    GridY[1,0:Y_vals.shape[0]] = Y_vals[:]


elif h == “N”:
    # dimensions grid size by maximum possible size of the grid in X
    # for only two polygons
    GridX = np.full((2,int(2*(LPS_S_phi0-LPS_S_Border)/ResY+
```

```
        int(2*(LatLon_E_Meridian-LatLon_W_Meridian)/ResX
        +4))),np.nan)
# dimensions grid size by maximum possible size of the grid in Y
# for only two polygons
GridY = np.full((2,int(2*(LPS_S_phi0-LPS_S_Border)/ResY+
              int(2*(LatLon_E_Meridian-LatLon_W_Meridian)/ResX+
              4))),np.nan)
# Poly 1
# Northern border

# Poly 1
# L - > R border
GridX_B = np.arange(LatLon_W_Meridian,0,ResX)[:,np.newaxis] #grid lines
GridX_B = np.append(GridX_B,0)

GridY_B = np.ones(GridX_B.shape[0]-1).T*LPS_N_Border
GridY_B = np.append(GridY_B,LPS_N_Border)

# S -> N
GridY_R = np.arange(LPS_N_Border,LPS_N_phi0,ResY)[:,np.newaxis] #grid lines
GridY_R = np.append(GridY_R,LPS_N_phi0)

GridX_R = np.ones(GridY_R.shape[0]-1).T*0
GridX_R = np.append(GridX_R,0)

# R -> L
GridX_T = np.arange(0,LatLon_W_Meridian,-ResX)[:,np.newaxis] #grid lines
GridX_T = np.append(GridX_T,LatLon_W_Meridian)

GridY_T = np.ones(GridX_T.shape[0]-1).T*LPS_N_phi0
GridY_T = np.append(GridY_T,LPS_N_phi0)

# N -> S
GridY_L = np.arange(LPS_N_phi0,LPS_N_Border,-ResY)[:,np.newaxis] #grid lines
GridY_L = np.append(GridY_L,LPS_N_Border)

GridX_L = np.ones(GridY_L.shape[0]-1).T*LatLon_W_Meridian
GridX_L = np.append(GridX_L,LatLon_W_Meridian)

# combine parts of polygon into list
X_vals = (GridX_B,GridX_R,GridX_T,GridX_L)
Y_vals = (GridY_B,GridY_R,GridY_T,GridY_L)

# concatenate values into single array
X_vals = np.concatenate(X_vals)
Y_vals = np.concatenate(Y_vals)

# combine grid values
GridX[0,0:X_vals.shape[0]] = X_vals[:]
GridY[0,0:Y_vals.shape[0]] = Y_vals[:]

# Poly 2
GridX_B = np.arange(0,LatLon_E_Meridian,ResX)[:,np.newaxis] #grid lines
GridX_B = np.append(GridX_B,LatLon_E_Meridian)

GridY_B = np.ones(GridX_B.shape[0]-1).T*LPS_N_Border
```

```
        GridY_B = np.append(GridY_B,LPS_N_Border)

        # S -> N
        GridY_R = np.arange(LPS_N_Border,LPS_N_phi0,ResY)[:,np.newaxis] #grid lines
        GridY_R = np.append(GridY_R,LPS_N_phi0)

        GridX_R = np.ones(GridY_R.shape[0]-1).T*LatLon_E_Meridian
        GridX_R = np.append(GridX_R,LatLon_E_Meridian)

        # R -> L
        GridX_T = np.arange(LatLon_E_Meridian,0,-ResX)[:,np.newaxis] #grid lines
        GridX_T = np.append(GridX_T,LatLon_W_Meridian)

        GridY_T = np.ones(GridX_T.shape[0]-1).T*LPS_N_phi0
        GridY_T = np.append(GridY_T,LPS_N_phi0)

        # N -> S
        GridY_L = np.arange(LPS_N_phi0,LPS_N_Border,-ResY)[:,np.newaxis] #grid lines
        GridY_L = np.append(GridY_L,LPS_N_Border)

        GridX_L = np.ones(GridY_L.shape[0]-1).T*0
        GridX_L = np.append(GridX_L,0)

        # combine grid values into a list
        X_vals = (GridX_B,GridX_R,GridX_T,GridX_L)
        X_vals = np.concatenate(X_vals)
        Y_vals = (GridY_B,GridY_R,GridY_T,GridY_L)
        Y_vals = np.concatenate(Y_vals)

        # assign grid values
        GridX[1,0:X_vals.shape[0]] = X_vals[:]
        GridY[1,0:Y_vals.shape[0]] = Y_vals[:]

    else:
        raise ValueError ("Hemisphere could not be determined for polar"
                          "LGRS global all grid.")

#  25km grid areas, in a single LPS zone in LPS Coordinates
elif Grid_Res == "25km":
    if h == "S":
        # generate empty arrays for processing
        GridX = np.full((2,int(2*(LPS_S_Border-LPS_S_phi0)/ResY+
                        int(LatLon_E_Meridian-LatLon_W_Meridian/ResX
                        +3))),np.nan)

        GridY = np.full((2,int(2*(LPS_S_Border-LPS_S_phi0)/ResY+
                         int(LatLon_E_Meridian-LatLon_W_Meridian/ResX+
                         3))),np.nan)

        # Determine LPS zone max and min values in meters
        # these are quadrant values, azimuth on the LPS
        # plain, not on the sphere.
        h,X_max,Y = toLPS(90,LPS_S_Border,trunc_val=0)
        h,X_min,Y = toLPS(-90,LPS_S_Border,trunc_val=0)
        h,X,Y_max = toLPS(-.0,LPS_S_Border,trunc_val=0)
        h,X,Y_min = toLPS(180,LPS_S_Border,trunc_val=0)
```

```
elif h == "N":
    # generate empty arrays for processing
    GridX = np.full((2,int(2*(LPS_N_phi0-LPS_N_Border)/ResY+
                int(LatLon_E_Meridian-LatLon_W_Meridian/ResX
                +3))),np.nan)

    GridY = np.full((2,int(2*(LPS_N_phi0-LPS_N_Border)/ResY+
                  int(LatLon_E_Meridian-LatLon_W_Meridian/ResX+
                  3))),np.nan)

    # Determine LPS zone max and min values in meters
    # these are quadrant values, azimuth on the LPS
    # plain, not on the sphere.
    h,X_max,Y = toLPS(90,LPS_N_Border,trunc_val=0)
    h,X_min,Y = toLPS(-90,LPS_N_Border,trunc_val=0)
    h,X,Y_max = toLPS(180,LPS_N_Border,trunc_val=0)
    h,X,Y_min = toLPS(-.0,LPS_N_Border,trunc_val=0)

else:
    raise ValueError ("LPS zone for Polar LGRS grid could not be determined")

# generate arrays. These values are in meters for the stereographic projection
# get min and max of grid areas to be generated
GridY_min = Y_min//25E3*25E3
GridY_max = (Y_max//25E3 + 1)*25E3

GridX_min = X_min//25E3*25E3
GridX_max = (X_max//25E3 + 1)*25E3

# generate 25km grid
StereoY = np.arange(np.abs(GridY_min),np.abs(GridY_max)+25E3,25E3)
StereoX = np.arange(np.abs(GridX_min),np.abs(GridX_max)+25E3,25E3)

# meshgrid for array of Xs and Ys
StereoY,StereoX = np.meshgrid(StereoX,StereoY)

# assign grid to geometry
LGRS_25km_geom_polar_area = assign_xy_grid_to_multipolygon(StereoY,StereoX)

# determine LTM bounding box for clipping
LPS_X,LPS_Y = generate_LPS_grid("global",h)

# convert coordinates to LPS values
for m in range(0,LPS_X.shape[0]):
    for n in range(0,LPS_X.shape[1]):
        _,LPS_X[m,n],LPS_Y[m,n] = toLPS(LPS_X[m,n],LPS_Y[m,n],trunc_val=0,process_errors=False)

# Assign LPS zone border to a polygon geometry
LPS_zone_clip_geom = assign_xy_to_polygon(LPS_X,LPS_Y)

# clip to bounding box
LGRS_area_clipped = clip_multipolygon_by_polygon(LGRS_25km_geom_polar_area,LPS_zone_clip_geom)

# extract X, Y positions from clipped region
GridX,GridY = extract_xy_from_multipolygon(LGRS_area_clipped)
```

```
# generate polar grid in grid sizes less than 25km
elif Grid_Res == “1km” or \
     Grid_Res == “100m” or \
     Grid_Res == “10m” or \
     Grid_Res == “1m”:

    # making a grid in multiple zones is not allowed for LGRS in
    # the LPS portions. Terminate if crosses into both polar
    # regions
    if h1!=h2:
        raise ValueError(“Polar LGRS cannot be generated in both”
                         “at the same time:\n Please double check”
                         “coordinates polar values”)

    # get grid dimensions in meters
    if Grid_Res == “1km”:
        grid_sizeY,grid_sizeX = 1E3,1E3
    elif Grid_Res == “100m”:
        grid_sizeY,grid_sizeX = 100,100
    elif Grid_Res == “10m”:
        grid_sizeY,grid_sizeX = 10,10
    elif Grid_Res == “1m”:
        grid_sizeY,grid_sizeX = 1,1

    if ulx == None or uly == None or lrx == None or lry == None:
        raise ValueError (“LGRS <1km polar grid requires an input box “
                          “provide ulx,uly,lrx,lry”)

    # generate single zone grid: extend to clipping border.
    # for simplicity the grids are contained within a whole zone

    # create box for new Coordinates to cross areas
    StereoX_min = grid_sizeX*(ulx//grid_sizeX)
    StereoX_max = grid_sizeX*(lrx//grid_sizeX+1)

    StereoY_min = grid_sizeY*(lry//grid_sizeY)
    StereoY_max = grid_sizeY*(uly//grid_sizeY + 1)

    # Generate arrays to specified size
    GridY = np.arange(StereoY_min,StereoY_max,grid_sizeY)
    GridX = np.arange(StereoX_min,StereoX_max,grid_sizeX)

    # append last value to guarantee endpoint
    GridY = np.append(GridY,StereoY_max)
    GridX = np.append(GridX,StereoX_max)

    # create grid
    GridX,GridY = np.meshgrid(GridX,GridY)

    # Clipping in polar regions takes a longtime
    # at high fidelity grids due to the number of polygons
    clip = False
    for i in range(0,GridY.shape[0]):
        for j in range(0,GridY.shape[1]):
            _,Y = toLatLon_Polar(GridX[i,j],GridY[i,j],h1,
```

```
                    eqs="Spherical",process_errors=False)
            # search to see if any point is outside of the LPS plot
            # area. Set clip to true if true

            if ( (h1 == "N" and Y < 80) or\
               (h1 == "S" and Y > -80) ):
               clip = True
               break

      #  A clipping operation to be applied to the
      # Polar LGRS region
      if clip:
         # assign grid LPS coordinates to polygon
         LGRS_grid_area = assign_xy_grid_to_multipolygon(GridX,GridY)

         # determine LPS bounding box for clipping
         LPS_X,LPS_Y = generate_LPS_grid("border",h1)

         # convert LPS border to LPS grid Coordinates
         for i in range(0,LPS_X.shape[1]):
            _,LPS_X[0,i],LPS_Y[0,i] = toLPS(LPS_X[0,i],LPS_Y[0,i],trunc_val=0,
                     eqs="Spherical",process_errors=True)

         # assign to geometry for clipping
         LPS_zone_clip_geom = assign_xy_to_polygon(LPS_X,LPS_Y)

         # clip to bounding box
         LGRS_grid_clipped = clip_multipolygon_by_polygon(LGRS_grid_area,LPS_zone_clip_geom)

         # extract X, Y positions from clipped region
         GridX,GridY = extract_xy_from_multipolygon(LGRS_grid_clipped)

      # if clipping is not needed
      else:
         # assign grid area to polygon
         LGRS_grid_area = assign_xy_grid_to_multipolygon(GridX,GridY)

         # export X and Y values from grid. This is completed for
         # proper formatting
         GridX,GridY = extract_xy_from_multipolygon(LGRS_grid_area)
   else:
      raise ValueError("PolarLGRS Generation Terminated: Grid type could"
            "not be determined. Please specify:\n"
            "\{global,global_all,25km,25km_all,1km,100m, "
            "10m, or 1m \}")

   # functionality to force the output grid into lunar LatLon
   # (not recommended as the grid will not be tied to a PCRS)

   if LatLon_override:
      if Grid_Res != "global_all":
         for i in range(0,GridX.shape[0]):
            for j in range(0,GridY.shape[1]):
               GridX[i,j],GridY[i,j]=toLatLon_Polar(GridX[i,j],
                                    GridY[i,j],
                                    h1,
```

```
                                    eqs="Spherical",
                                    process_errors=True)
        else:
            raise ValueError("LatLon Override could not be applied as "
                             "grids are already in a LatLon format:\n"
                             "Set latlon_override to False to continue")
    return GridX, GridY

# ============================================================================
# SECTION 3: MAIN PROGRAM

# SECTION 3.0: Initialize and run main program
# SECTION 3.1: Determine if input arguments are correct
# SECTION 3.2: Read in input arguments center x center y or corners,
# SECTION 3.3: Converting Input Coordinates to Common Grid Format.
# SECTION 3.4: Generate the Correct Grid to the desired resolution
# SECTION 3.5: Convert grid Coordinates to correct output format.
# SECTION 3.6: Determine grid cell names in specified output format
# SECTION 3.7: Assign output WKT1 Projection string for writing file
# SECTION 3.8: Write grid to Esri shapefile and format output names


# ----------------------------------------------------------------------
# SECTION 3.0: Initialize and run main program

def main(form_in, form_out, Grid_Res,
                    multigrid=False,
                    attribute=True,
                    overwrite=False,
                    output_file_form="shp",
                    info=True,
                    LatLon_override=False):

    """
    Run main program and generate arrays. Descriptions
    provided for each step

    Input Parameters:
    (str) form_in - Coordinate input format
    (str) form_out - Coordinate output format
    (str) Grid_Res - type of grid to be generated
    (str) units - specify units (placed for later code improvements)
    (bool) multigrid - processing flag that enables an LTM grid to
                be generated across multiple zones and hemispheres.
                grid is forced in LatLon
    (bool) attribute - processing flag that enables export of a
                a shapefiles attribute table
    (bool) overwrite - processing flag that will remove a shapefile if
                it already exists.
    (str) output_file_form - output file format. Current support
                    includes Esri shapefiles
    (bool) info - processing flag prints processing information to the
                user.
    (bool) LatLon_override - Force the grids to output in lunar Latlon
                    only applies to LGRS and LGRS in ACC format
                    NOTE: these projections must still be used
```

```
                in LPS or LTM to ensure correct spatial
                dimensions.

    Correct format in values for form_in
    Lunar Transverse Mercator - (LTM)
    Lunar Polar Stereographic - (LPS)
    Lunar grid Reference System - (LGRS)
    Polar LGRS - (PolarLGRS)
    LGRS in Artemis Condensed Coordinate (ACC) Format - (LGRS_ACC)
    Polar LGRS in ACC Format - (PolarLGRS_ACC)

    Correct format out values for form_out
    LTM
    LPS
    LGRS
    LGRS_ACC
    ACC
    PolarLGRS
    PolarLGRS_ACC
    LGRS in ACC Format truncated to 10m without global
            reference - (PolarLGRS_ACC)
    Polar LGRS in ACC Format truncated to 10m without global
            reference - (PolarACC)

    Types of grids for Grid_Res
    global_all - All global areas
    global - global areas within a single LTM or LPS zone
    border - LTM or LPS zone borders
    25km - LGRS rectangular grid at a 25km resolution
    25km_all - Generate all LGRS 25km grid areas
    1km - LGRS rectangular grid at a 1km resolution
    100m - LGRS rectangular grid at a 100m resolution
    10m - LGRS rectangular grid at a 10m resolution
    1m - LGRS rectangular grid at a 1m resolution

    Returns:
    None

    Exports:
    (file) - Esri shapefile (shp,shx,dbf, and prj)
    (folder) - Esri shapefile containing folder
    “”””

    # using Coordinate Conversion program, initialize its Globals
    # for calculations
    initialize_LGRS_function_globals()

# ---------------------------------------------------------------------
# SECTION 3.1: Determine if input arguments are correct
#       grid and grid resolution are checked.
#       We have 8 different coordinate formats and 7 different grid sizes
#       Not all work together. This section determines if GridRes and
#       form_in are comparable.


    try:
```

```
#    Acceptable coordinate system and grid resolution combinations
    if not ( ( form_out == “LTM” and Grid_Res == “global_all” ) or \
        ( form_out == “LTM” and Grid_Res == “global” ) or \
        ( form_out == “LTM” and Grid_Res == “border” ) or \
        ( form_out == “LPS” and Grid_Res == “global” ) or \
        ( form_out == “LPS” and Grid_Res == “border” ) or \
        ( form_out == “LGRS” and Grid_Res == “global” ) or \
        ( form_out == “LGRS” and Grid_Res == “global_all” ) or \
        ( form_out == “LGRS” and Grid_Res == “border” ) or \
        ( form_out == “LGRS” and Grid_Res == “25km” ) or \
        ( form_out == “LGRS” and Grid_Res == “25km_all” ) or \
        ( form_out == “LGRS” and Grid_Res == “1km” ) or \
        ( form_out == “LGRS” and Grid_Res == “100m” ) or \
        ( form_out == “LGRS” and Grid_Res == “10m” ) or \
        ( form_out == “LGRS” and Grid_Res == “1m” ) or \
        ( form_out == “PolarLGRS” and Grid_Res == “global” ) or \
        ( form_out == “PolarLGRS” and Grid_Res == “global_all” ) or \
        ( form_out == “PolarLGRS” and Grid_Res == “25km” ) or \
        ( form_out == “PolarLGRS” and Grid_Res == “1km” ) or \
        ( form_out == “PolarLGRS” and Grid_Res == “100m” ) or \
        ( form_out == “PolarLGRS” and Grid_Res == “10m” ) or \
        ( form_out == “PolarLGRS” and Grid_Res == “1m” ) or \
        ( form_out == “LGRS_ACC” and Grid_Res == “1km” ) or \
        ( form_out == “LGRS_ACC” and Grid_Res == “100m” ) or \
        ( form_out == “LGRS_ACC” and Grid_Res == “10m” ) or \
        ( form_out == “LGRS_ACC” and Grid_Res == “1m” ) or \
        ( form_out == “PolarLGRS_ACC” and Grid_Res == “1km” ) or \
        ( form_out == “PolarLGRS_ACC” and Grid_Res == “100m” ) or \
        ( form_out == “PolarLGRS_ACC” and Grid_Res == “10m” ) or \
        ( form_out == “PolarLGRS_ACC” and Grid_Res == “1m” ) or \
        ( form_out == “ACC” and Grid_Res == “1km” ) or \
        ( form_out == “ACC” and Grid_Res == “100m” ) or \
        ( form_out == “ACC” and Grid_Res == “10m” ) or \
        ( form_out == “PolarACC” and Grid_Res == “1km” ) or \
        ( form_out == “PolarACC” and Grid_Res == “100m” ) or \
        ( form_out == “PolarACC” and Grid_Res == “10m” ) ):
        raise ValueError(“Output coordinate system and Grid Resolution cannot “
                “be plotted.\n”
                “Please provide one of the following:\n:”
                “form_out: LTM | grid_res: global_all, or\n”
                “form_out: LTM | grid_res: global, or \n”
                “form_out: LTM | grid_res: border, or\n”
                “form_out: LPS | grid_res: border, or\n”
                “form_out: LPS | grid_res: global, or\n”
                “form_out: LGRS | grid_res: global, or\n”
                “form_out: LGRS | grid_res: global_all, or\n”
                “form_out: LGRS | grid_res: 25km, or\n”
                “form_out: LGRS | grid_res: 25km_all, or\n”
                “form_out: LGRS | grid_res: 1km, or\n”
                “form_out: LGRS | grid_res: 100m, or\n”
                “form_out: LGRS | grid_res: 10m, or\n”
                “form_out: LGRS | grid_res: 1m, or\n”
                “form_out: PolarLGRS | grid_res: global, or\n”
                “form_out: PolarLGRS | grid_res: global_all, or\n”
                “form_out: PolarLGRS | grid_res: 25km, or\n”
                “form_out: PolarLGRS | grid_res: 1km, or\n”
```

```
                "form_out: PolarLGRS | grid_res: 100m, or\n"
                "form_out: PolarLGRS | grid_res: 10m, or\n"
                "form_out: PolarLGRS | grid_res: 1m, or\n"
                "form_out: LGRS_ACC | grid_res: 1km, or\n"
                "form_out: LGRS_ACC | grid_res: 100m, or\n"
                "form_out: LGRS_ACC | grid_res: 10m, or\n"
                "form_out: LGRS_ACC | grid_res: 1m, or\n"
                "form_out: PolarLGRS_ACC | grid_res: 1km, or\n"
                "form_out: PolarLGRS_ACC | grid_res: 100m, or\n"
                "form_out: PolarLGRS_ACC | grid_res: 10m, or\n"
                "form_out: PolarLGRS_ACC | grid_res: 1m,  or\n"
                "form_out: ACC | Grid_Res: 1km, or \n"
                "form_out: ACC | Grid_Res: 100m, or \n"
                "form_out: ACC | Grid_Res: 10m, or \n"
                "form_out: PolarACC | Grid_Res: 1km, or \n"
                "form_out: PolarACC | Grid_Res: 100m, or \n"
                "form_out: PolarACC | Grid_Res: 10m")

    # this checks to make sure the right input coordinates can
    # be used to create a grid
    if not ( ( form_in == "LTM" and form_out == "LTM" ) or \
        ( form_in == "LTM" and form_out == "LGRS" ) or \
        ( form_in == "LTM" and form_out == "LGRS_ACC" ) or \
        ( form_in == "LTM" and form_out == "ACC" ) or \
        ( form_in == "LPS" and form_out == "LPS" ) or \
        ( form_in == "LPS" and form_out == "PolarLGRS" ) or \
        ( form_in == "LPS" and form_out == "PolarLGRS_ACC" ) or \
        ( form_in == "LPS" and form_out == "PolarACC" ) or \
        ( form_in == "LGRS" and form_out == "LGRS" ) or \
        ( form_in == "LGRS" and form_out == "LGRS_ACC" ) or \
        ( form_in == "LGRS" and form_out == "ACC" ) or \
        ( form_in == "PolarLGRS" and form_out == "PolarLGRS" ) or \
        ( form_in == "PolarLGRS" and form_out == "PolarLGRS_ACC" ) or \
        ( form_in == "PolarLGRS" and form_out == "PolarACC" ) or \
        ( form_in == "LGRS_ACC" and form_out == "LGRS_ACC" ) or \
        ( form_in == "LGRS_ACC" and form_out == "ACC" ) or \
        ( form_in == "PolarLGRS_ACC" and form_out == "PolarLGRS_ACC" ) or \
        ( form_in == "PolarLGRS_ACC" and form_out == "PolarACC" ) or \
        ( form_in == "LatLon" and form_out == "LTM" ) or \
        ( form_in == "LatLon" and form_out == "LGRS" ) or \
        ( form_in == "LatLon" and form_out == "LGRS_ACC" ) or \
        ( form_in == "LatLon" and form_out == "ACC" ) or \
        ( form_in == "PolarLatLon" and form_out == "LPS" ) or \
        ( form_in == "PolarLatLon" and form_out == "PolarLGRS" ) or \
        ( form_in == "PolarLatLon" and form_out == "PolarLGRS_ACC" ) or \
        ( form_in == "PolarLatLon" and form_out == "PolarACC" ) ):
       raise ValueError("Input coordinate system and Output coordinate system "
            "cannot be processed. Please provide one of the following:\n"
            "form_in: LTM | form_out: LTM, or\n"
            "form_in: LTM | form_out: LGRS, or\n"
            "form_in: LTM | form_out: LGRS_ACC, or\n"
            "form_in: LTM | form_out: ACC, or\n"
            "form_in: LPS | form_out: LPS, or\n"
            "form_in: LPS | form_out: PolarLGRS, or\n"
            "form_in: LPS | form_out: PolarLGRS_ACC, or\n"
            "form_in: LPS | form_out: PolarACC, or\n"
```

```
                "form_in: LGRS | form_out: LGRS, or\n"
                "form_in: LGRS | form_out: LGRS_ACC, or\n"
                "form_in: LGRS | form_out: ACC, or\n"
                "form_in: PolarLGRS | form_out: PolarLGRS, or\n"
                "form_in: PolarLGRS | form_out: PolarLGRS_ACC, or\n"
                "form_in: PolarLGRS | form_out: PolarACC, or\n"
                "form_in: LGRS_ACC | form_out: LGRS_ACC, or\n"
                "form_in: LGRS_ACC | form_out: ACC, or\n"
                "form_in: PolarLGRS_ACC | form_out: PolarLGRS_ACC, or\n"
                "form_in: PolarLGRS_ACC | form_out: PolarACC, or\n"
                "form_in: LatLon | form_out: LTM, or\n"
                "form_in: LatLon | form_out: LGRS, or\n"
                "form_in: LatLon | form_out: LGRS_ACC, or\n"
                "form_in: LatLon | form_out: ACC, or\n"
                "form_in: PolarLatLon | form_out: LPS, or\n"
                "form_in: PolarLatLon | form_out: PolarLGRS, or\n"
                "form_in: PolarLatLon | form_out: PolarLGRS_ACC, or\n"
                "form_in: PolarLatLon | form_out: PolarACCPolarACC.")
    # print faults
    except ValueError as e:
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        print(exc_type, fname, exc_tb.tb_lineno)
        print(e)
        exit(1)

# ----------------------------------------------------------------------
# SECTION 3.2: Read in input arguments center x center y or corners,
#              (ulx uly lrx lry) to form a bounding  box to dimension
#              the grid in


    try:
        # Planetocentric Latitude and Longitude
        if form_in == "LatLon":
            if Grid_Res == "global_all" or Grid_Res == "25km_all" :
                h,z = None,None
            elif Grid_Res not in grid_format[5:]: # 25km or greater
                if len(sys.argv)-1 == 5:
                    cent_x=float(sys.argv[4])
                    cent_y=float(sys.argv[5])
                else:
                    raise IndexError("Operation aborted: Number of inputs not correct."
                                     "for grid >=25km: coordinate: Center_X Center_Y")
            elif Grid_Res in grid_format:
                if len(sys.argv)-1 == 7:
                    ulx = float(sys.argv[4])
                    uly = float(sys.argv[5])
                    lrx = float(sys.argv[6])
                    lry = float(sys.argv[7])
                else:
                    raise IndexError("Operation aborted: Number of inputs not correct."
                        "for grid <25km LatLon:\nulx Lon\nuly lat\nlrx lon\nlry lat")
            else:
                raise IndexError("Operation aborted: Number of inputs not correct."
                        "Input for LatLon:\nulx Lat\nuly Lon\nlrx Lat\nlry Lon"
```

```
            "or the single coordinate: Center_X Center_Y")

    elif form_in == "PolarLatLon":
        if Grid_Res not in grid_format[5:]: # 25km or greater
            if len(sys.argv)-1 == 5:
                cent_x = float(sys.argv[4])
                cent_y = float(sys.argv[5])
            else:
                raise IndexError("Operation aborted: Number of inputs not correct."
                        "for grid >=25km: coordinate: Center_X Center_Y")
        elif Grid_Res in grid_format:
            if len(sys.argv)-1 == 7:
                ulx = float(sys.argv[4])
                uly = float(sys.argv[5])
                lrx = float(sys.argv[6])
                lry = float(sys.argv[7])
            else:
                raise IndexError("Operation aborted: Number of inputs not correct."
                    "for grid <25km LatLon:\nulx Lon\nuly lat\nlrx lon\nlry lat")
        else:
            raise IndexError("Operation aborted: Number of inputs not correct."
                    "Input for LatLon:\nulx Lat\nuly Lon\nlrx Lat\nlry Lon"
                    "or the single coordinate: Center_X Center_Y")
    # Lunar Transverse Mercator
    elif form_in == "LTM":
        if Grid_Res == "global_all" or Grid_Res == "25km_all" :
            h,z = None,None
        elif Grid_Res not in grid_format[5:]:
            if len(sys.argv)-1 == 5:
                z = int(sys.argv[4])
                h = str(sys.argv[5])

            elif len(sys.argv)-1 == 4:
                try:
                    z,h = list(re.findall(LTM_pattern,sys.argv[4])[0])
                    z = int(z)
                    h = str(h)

                except:
                    raise IndexError("Operation aborted: Number of inputs not correct."
                        "\nInput for LTM: zone\{1-45\}, hemisphere \"S\" | \"N\"")
            else:
                raise IndexError("Operation aborted: Number of inputs not correct."
                        "\nInput for LTM: zone\{1-45\}, hemisphere \"S\" | \"N\"")
        elif Grid_Res in grid_format[5:]:
                if len(sys.argv)-1 == 5:
                    try:
                        z1,h1,ulx,_,uly,_ = list(re.findall(LTM_pattern_full,sys.argv[4])[0])
                        z2,h2,lrx,_,lry,_ = list(re.findall(LTM_pattern_full,sys.argv[5])[0])

                        z1,h1,ulx,uly = int(z1),str(h1),float(ulx),float(uly)
                        z2,h2,lrx,lry = int(z2),str(h2),float(lrx),float(lry)
                    except:
                        raise IndexError("Operation aborted: LTM Coordinates could not be"
                                " converted to proper format correctly.")
                else:
```

```
            raise IndexError("Operation aborted: Number of inputs not correct."
                "\nInput for LTM: zone1\{1-45\}, hemisphere1\"S\"| \"N\" Easting 1, Northing 1"
                "\nInput for LTM: zone2\{1-45\}, hemisphere2\"S\"| \"N\" Easting 2, Northing 2")
    else:
        raise IndexError("Operation aborted: Input Coordinates could not be read")
    # convert to proper data formats

# Lunar polar stereographic Coordinates
elif form_in == "LPS":

    if Grid_Res not in grid_format[5:]:
        if len(sys.argv)-1 == 4:
            try:
                h = str(sys.argv[4])
            except:
                raise IndexError("Operation aborted: LPS Coordinates could not be"
                    " converted to proper format correctly.")

    elif Grid_Res in grid_format[5:]:
        if len(sys.argv)-1 == 5:
            try:
                h1,ulx,_,uly,_ = list(re.findall(LPS_pattern_full,sys.argv[4])[0])
                h2,lrx,_,lry,_ = list(re.findall(LPS_pattern_full,sys.argv[5])[0])
            except:
                raise IndexError("Operation aborted: LPS Coordinates could not be"
                    " converted to proper format correctly. Check input arguments")

            h1,ulx,uly = str(h1),float(ulx),float(uly)
            h2,lrx,lry = str(h2),float(lrx),float(lry)

        else:
            raise IndexError("Operation aborted: Number of inputs not correct."
                "\nInput for LPS: hemisphere1\"S\"| \"N\" Easting 1, Northing 1"
                "\nInput for LPS: hemisphere2\"S\"| \"N\" Easting 2, Northing 2")

# Lunar Grid Reference System in LTM regions
elif form_in == "LGRS":
    if Grid_Res  ==  "global_all" or Grid_Res == "25km_all":
        z,h = None,None

    elif Grid_Res not in grid_format[5:]: # 25km or greater
        if len(sys.argv)-1 == 5: #single zone area
            z = sys.argv[4]
            h = sys.argv[5]

        elif len(sys.argv)-1 == 4: # bounding box in condensed Coordinates
            try:
                z,h = list(re.findall(LTM_pattern,sys.argv[4])[0])

            except:

                raise IndexError("Operation aborted: Number of inputs not correct."
                    "Input for LGRS coarse grid (>=25km):\n Zone \{1-45\}"
                    " hemisphere \"S\" | \"N\"")
        else:
            raise IndexError("Operation aborted: Number of inputs not correct."
```

```
                    "Maybe try a smaller grid side.\n"
                    "Input for LGRS coarse grid (>=25km):\n Zone \{1-45\}"
                    " hemisphere \"S\" | \"N\"")
        # convert to proper data formats
        try:
            z = int(z)
            h = str(h)
        except:
            raise IndexError("Operation aborted: LGRS Coordinates could not be"
                    " converted to proper format correctly.")

    elif Grid_Res in grid_format:
        if len(sys.argv)-1 == 15:

            lonBand1 = sys.argv[4]
            latBand1 = sys.argv[5]
            e25k1 = sys.argv[6]
            n25k1 = sys.argv[7]
            E1 = sys.argv[8]
            N1 = sys.argv[9]

            lonBand2 = sys.argv[10]
            latBand2 = sys.argv[11]
            e25k2 = sys.argv[12]
            n25k2 = sys.argv[13]
            E2 = sys.argv[14]
            N2 = sys.argv[15]

        elif len(sys.argv)-1 == 5: # bounding box in condensed Coordinates
            try:
                lonBand1,latBand1,e25k1,n25k1,E1,N1=list(re.findall(LGRS_pattern,sys.argv[4])[0])
                lonBand2,latBand2,e25k2,n25k2,E2,N2=list(re.findall(LGRS_pattern,sys.argv[5])[0])
            except:
                raise IndexError("Operation aborted: Number of inputs not correct."
                        "\nInput LGRS fine grids (less than 25km):LonBandlatBande25kn25kEN"
                        " LR:LonBandlatBande25kn25kEN"
                        "\nor in an uncondensed format:"
                        "\nLGRS UL:LonBand latBand e25k n25k E N LR:LonBand latBand e25k n25k E N")
        else:
            raise IndexError("Operation aborted: Number of inputs not correct."
                        "\nInput LGRS fine gris (less than 25km):LonBandlatBande25kn25kEN"
                        " LR:LonBandlatBande25kn25kEN"
                        "\nor in an uncondensed format:"
                        "\nLGRS UL:LonBand latBand e25k n25k E N LR:LonBand latBand e25k n25k E N")

        # convert to proper data formats
        try:
            lonBand1 = int(lonBand1)
            latBand1 = str(latBand1)
            e25k1 = str(e25k1)
            n25k1 = str(n25k1)
            E1 = float(E1)
            N1 = float(N1)

            lonBand2 = int(lonBand2)
            latBand2 = str(latBand2)
```

```
            e25k2 = str(e25k2)
            n25k2 = str(n25k2)
            E2 = float(E2)
            N2 = float(N2)
        except:
            raise IndexError("Operation aborted: LGRS Coordinates could not be"
                " converted to proper format correctly.")
    else:
        raise IndexError("Operation aborted: LGRS grid size could not be determined"
                    ". Please select one of the following:\n{}".format(grid_format_str))

# Lunar Grid Reference System in LPS regions
elif form_in == "PolarLGRS":
    if Grid_Res not in grid_format[5:]: # 25km or greater

        if len(sys.argv)-1 == 4:
            h = str(sys.argv[4])

        else:
            raise IndexError("Operation aborted: LGRS grid size could not be determined"
                "Input for Polar LGRS coarse grid (>=25km): }"
                "hemisphere \"S\" | \"N\"")

    elif Grid_Res in grid_format:
        if len(sys.argv)-1 == 13:

            lonBand1 = sys.argv[4]
            e25k1 = sys.argv[5]
            n25k1 = sys.argv[6]
            E1 = sys.argv[7]
            N1 = sys.argv[8]

            lonBand2 = sys.argv[9]
            e25k2 = sys.argv[10]
            n25k2 = sys.argv[11]
            E2 = sys.argv[12]
            N2 = sys.argv[13]

        elif len(sys.argv)-1 == 5:
            try:
                lonBand1,e25k1,n25k1,E1,N1 = list(re.findall(PolarLGRS_pattern,sys.argv[4])[0])
                lonBand2,e25k2,n25k2,E2,N2 = list(re.findall(PolarLGRS_pattern,sys.argv[5])[0])
            except:
                raise IndexError("Operation aborted: Number of inputs not correct."
                "\nInput for condensed PolarLGRS UL:LonBande25kn25kEN LR:LonBande25kn25kEN"
                "\nor"
                "\nPolarLGRS UL:LonBand e25k n25k E N LR:LonBand e25k n25k E N")

        else:
            raise IndexError("Operation aborted: Number of inputs not correct."
                "\nInput for condensed PolarLGRS UL:LonBande25kn25kEN LR:LonBande25kn25kEN"
                "\nor"
                "\nPolarLGRS UL:LonBand e25k n25k E N LR:LonBand e25k n25k E N")
        # convert to proper data formats
        try:
            lonBand1 = str(lonBand1)
```

```
            e25k1 = str(e25k1)
            n25k1 = str(n25k1)
            E1 = float(E1)
            N1 = float(N1)

            lonBand2 = str(lonBand2)
            e25k2 = str(e25k2)
            n25k2 = str(n25k2)
            E2 = float(E2)
            N2 = float(N2)
         except:
            raise IndexError("Operation aborted: Polar LGRS Coordinates could not be"
               " converted to proper format correctly.")


      else:
         raise IndexError("Operation aborted: PolarLGRS grid size could not be determined. "
                  "Please select one of the following:\n{}".format(grid_format_str))

   # Lunar Grid Reference System in ACC format in LTM regions
   elif form_in == "LGRS_ACC":

      if Grid_Res in grid_format[5:]: # any <= 25km or smaller
         if len(sys.argv)-1 == 19:

            lonBand1 = int(sys.argv[4])
            latBand1 = str(sys.argv[5])
            e25k1 = str(sys.argv[6])
            n25k1 = str(sys.argv[7])
            e1k1 = str(sys.argv[8])
            E1 = float(sys.argv[9])
            n1k1 = str(sys.argv[10])
            N1 = float(sys.argv[11])

            lonBand2 = int(sys.argv[12])
            latBand2 = str(sys.argv[13])
            e25k2 = str(sys.argv[14])
            n25k2 = str(sys.argv[15])
            e1k2 = str(sys.argv[16])
            E2 = float(sys.argv[17])
            n1k2 = str(sys.argv[18])
            N2 = float(sys.argv[19])

         elif len(sys.argv)-1 == 5:
            try:
               lonBand1,latBand1,e25k1,n25k1,e1k1,E1,n1k1,N1 = list(re.findall(LGRS_ACC_pattern,sys.argv[4])[0])
               lonBand2,latBand2,e25k2,n25k2,e1k2,E2,n1k2,N2 = list(re.findall(LGRS_ACC_pattern,sys.argv[5])[0])

            except:
               raise IndexError("Operation aborted: Number of inputs not correct."
                  "\nInput for condensed LGRS ACC UL:LonBandlatBande25km25k1ekmE1nkmN LR:LonBandlatBande
25km25km1ekmE1nkmN"
                  "\nor"
                  "\nLGRS ACC UL:LonBand latBand e25km 25km 1ekm E 1nkm N LR:LonBand latBand e25km 25km
1ekm E1 1nkm N")
         else:
```

```
            raise IndexError("Operation aborted: Number of inputs not correct."
                             "\nInput for condensed LGRS ACC UL:LonBandlatBande25km25k1ekmE1nkmN LR:LonBandlatB
ande25km25km1ekmE1nkmN"
                             "\nor"
                             "\nLGRS ACC UL:LonBand latBand e25km 25km 1ekm E 1nkm N LR:LonBand latBand e25km
25km 1ekm E1 1nkm N")

            # convert to proper data formats
            try:
               lonBand1 = int(lonBand1)
               latBand1 = str(latBand1)
               e25k1 = str(e25k1)
               n25k1 = str(n25k1)
               e1k1 = str(e1k1)
               E1 = float(E1)
               n1k1 = str(n1k1)
               N1 = float(N1)

               lonBand2 = int(lonBand2)
               latBand2 = str(latBand2)
               e25k2 = str(e25k2)
               n25k2 = str(n25k2)
               e1k2 = str(e1k2)
               E2 = float(E2)
               n1k2 = str(n1k2)
               N2 = float(N2)
            except:
               raise IndexError("Operation aborted: LGRS_ACC Coordinates could not be"
                  " converted to proper format correctly.")

         else:
            raise IndexError("Operation aborted: Grid size for LGRS in ACC must be <=25km"
                             "Correct grid size could not be determined. "
                             "Please select one of the following:\n{}".format(grid_format[5:]))

      # Lunar Grid Reference System in ACC format in LTM regions
      elif form_in == "PolarLGRS_ACC":
         if Grid_Res in grid_format[5:]: # any <= 25km or smaller
            if len(sys.argv)-1 == 17:
               lonBand1 = sys.argv[4]
               e25k1 = sys.argv[5]
               n25k1 = sys.argv[6]
               e1k1 = sys.argv[7]
               E1 = sys.argv[8]
               n1k1 = sys.argv[9]
               N1 = sys.argv[10]

               lonBand2 = sys.argv[11]
               e25k2 = sys.argv[12]
               n25k2 = sys.argv[13]
               e1k2 = sys.argv[14]
               E2 = sys.argv[15]
               n1k2 = sys.argv[16]
               N2 = sys.argv[17]

            elif len(sys.argv)-1 == 5:
```

```
                try:
                    lonBand1,e25k1,n25k1,e1k1,E1,n1k1,N1 = list(re.findall(PolarLGRS_ACC_pattern,sys.argv[4])[0])
                    lonBand2,e25k2,n25k2,e1k2,E2,n1k2,N2 = list(re.findall(PolarLGRS_ACC_pattern,sys.argv[5])[0])
                except:
                    raise IndexError("Operation aborted: Number of inputs not correct."
                            "\nInput for condensed Polar LGRS ACC UL:LonBande25km25k1ekmE1nkmN LR:LonBandlatBande
25km25km1ekmE1nkmN"
                            "\nor"
                            "\nPolar LGRS ACC UL:LonBand e25km 25km 1ekm E 1nkm N LR:LonBand e25km 25km 1ekm E1
1nkm N")
            else:
                raise IndexError("Operation aborted: Number of inputs not correct."
                            "\nInput for condensed Polar LGRS ACC UL:LonBande25km25k1ekmE1nkmN LR:LonBandlatBan
de25km25km1ekmE1nkmN"
                            "\nor"
                            "\nPolar LGRS ACC UL:LonBand e25km 25km 1ekm E 1nkm N LR:LonBand e25km 25km 1ekm
E1 1nkm N")

            # convert to proper data formats
            try:
                lonBand1 = str(lonBand1)
                e25k1 = str(e25k1)
                n25k1 = str(n25k1)
                e1k1 = str(e1k1)
                E1 = float(E1)
                n1k1 = str(n1k1)
                N1 = float(N1)

                lonBand2 = str(lonBand2)
                e25k2 = str(e25k2)
                n25k2 = str(n25k2)
                e1k2 = str(e1k2)
                E2 = float(E2)
                n1k2 = str(n1k2)
                N2 = float(N2)
            except:
                raise IndexError("Operation aborted: Polar LGRS ACC Coordinates could not be"
                    " converted to proper format correctly.")

        else:
            raise IndexError("Operation aborted: Grid size for LGRS in ACC must be <=25km"
                        "Correct grid size could not be determined"
                        ". Please select one of the following:\n{}".format(grid_format[5:]))

    else:
        raise IndexError("Number of inputs not correct."
                    "Input 1. Coordinate import format"
                    "2. Coordinate output format Form_out "
                    "and the  corner points for a grid bounding"
                    "box: ul(X|Y) lr(X|Y)")

except IndexError as e:
    exc_type, exc_obj, exc_tb = sys.exc_info()
    fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
    print(exc_type, fname, exc_tb.tb_lineno)
    print(e)
```

```
    exit(1)

# ---------------------------------------------------------------------
# SECTION 3.3: Converting Input Coordinates to Common Grid Format.
#         LPS or LTM format is chosen for simplicity. Note these are
#         inverse conversions. Any truncation prior to this program will
#         cause rounding of the original position.
#         Program ends by checking to make sure that if two sets of
#         Coordinates are provided, that that conform to having the
#         upper left coordinate be coordinate 1 and the lower right
#         to be coordinate 2.


  try:
    if form_in == "LatLon":
      if Grid_Res == "global_all" or Grid_Res == "25km_all":
        z,h = None,None
      elif Grid_Res not in grid_format[5:]: # 25km or greater
        z,h,E,N = toLTM(cent_x,cent_y,zone=None,trunc_val=0,process_errors=True)

      elif Grid_Res in grid_format:
        z1,h1,ulx,uly = toLTM(ulx,uly,zone=None,trunc_val=0,process_errors=True)
        z2,h2,lrx,lry = toLTM(lrx,lry,zone=None,trunc_val=0,process_errors=True)
      else:
        raise ValueError("Operation aborted: Conversion Method from LatLon to common format failed.")

    elif form_in == "PolarLatLon":
      if Grid_Res not in grid_format[5:]: # 25km or greater
        h,E,N = toLPS(cent_x,cent_y,trunc_val=0,process_errors=True)

      elif Grid_Res in grid_format:
        h1,ulx,uly = toLPS(ulx,uly,trunc_val=0,process_errors=True)
        h2,lrx,lry = toLPS(lrx,lry,trunc_val=0,process_errors=True)
      else:
        raise ValueError("Operation aborted:  Conversion Method from PolarLatLon to common format failed.")

    elif form_in == "LGRS":
      if Grid_Res not in grid_format[5:]: # 25km or greater
        z = z
        h = h
      elif Grid_Res in grid_format:
        z1,h1,ulx,uly = LGRStoLTM(latBand1,lonBand1,e25k1,n25k1,E1,N1)
        z2,h2,lrx,lry = LGRStoLTM(latBand2,lonBand2,e25k2,n25k2,E2,N2)
      else:
        raise ValueError("Operation aborted: Conversion Method from LGRS to common format failed.")

    elif form_in == "PolarLGRS":
      if Grid_Res not in grid_format[5:]: # 25km or greater
        h = h
      elif Grid_Res in grid_format:
        h1,ulx,uly = LGRStoLPS(lonBand1,e25k1,n25k1,E1,N1,process_errors=True)
        h2,lrx,lry=LGRStoLPS(lonBand2,e25k2,n25k2,E2,N2,process_errors=True)
      else:
        raise ValueError("Operation aborted: Conversion Method from Polar LGRS to common format failed.")

    elif form_in == "LGRS_ACC":
```

```
            if Grid_Res in grid_format:
                z1,h1,ulx,uly = LGRS_ACCtoLTM(latBand1,lonBand1,e25k1,n25k1,e1k1,E1,n1k1,N1,ACC=False,proc
ess_errors=True)
                z2,h2,lrx,lry = LGRS_ACCtoLTM(latBand2,lonBand2,e25k2,n25k2,e1k2,E2,n1k2,N2,ACC=False,proc
ess_errors=True)
            else:
                raise ValueError("Operation aborted: Conversion Method from Polar LGRS_ACC to common format failed.")

        elif form_in == "PolarLGRS_ACC":
            if Grid_Res in grid_format[5:]:
                h1,ulx,uly = LGRS_ACCtoLPS(lonBand1,e25k1,n25k1,e1k1,E1,n1k1,N1,ACC=False,process_errors=True)
                h2,lrx,lry = LGRS_ACCtoLPS(lonBand2,e25k2,n25k2,e1k2,E2,n1k2,N2,ACC=False,process_errors=True)
            else:
                raise ValueError("Operation aborted: Conversion Method from Polar LGRS_ACC to common format failed.")
        elif form_in == "LPS":
            pass

        elif form_in == "LTM":
            if Grid_Res == "global_all" or Grid_Res == "25km_all":
                z,h = None,None
            else:
                pass
        else:
            raise ValueError("Operation aborted: Conversion Method To Common input format failed."
                        "Check input variables")

        # check Coordinates to make sure bounding box is formatted correctly
        if Grid_Res in grid_format[5:]:

            # test equatorial areas
            if ( ( form_in == "LTM") or \
                ( form_in == "LatLon") or \
                ( form_in == "LGRS") or \
                ( form_in == "LGRS_ACC" ) ):

                # convert to LatLon to test box area for all input formats
                # and across multiple zones
                ulx_test,uly_test=toLatLon(ulx,uly,z1,h1,process_errors=True)
                lrx_test,lry_test=toLatLon(lrx,lry,z2,h2,process_errors=True)

                # check to make sure grid area is specified correctly
                if ulx_test > lrx_test:
                    raise ValueError ("Grid Easting of ULX greater than LRX")
                elif ulx_test == lrx_test:
                    raise ValueError ("Grid ULX and LRX Eastings are equal, no area to plot")
                elif uly_test < lry_test:
                    raise ValueError ("Grid ULY northing less than LRY.")
                elif uly_test == lry_test:
                    raise ValueError ("Grid ULY and LRY are equal, no area to plot")
            # test polar areas
            elif ( ( form_in == "LPS") or \
                ( form_in == "PolarLatLon") or \
                ( form_in == "PolarLGRS") or \
                ( form_in == "PolarLGRS_ACC" ) ):

                # no multiplots for polar regions, conversion to LatLon
```

```
            # not needed
            if ulx > lrx:
               raise ValueError ("Grid Easting of ULX greater than LRX")
            elif ulx == lrx:
               raise ValueError ("Grid ULX and LRX Eastings are equal, no area to plot")
            elif uly < lry:
               raise ValueError ("Grid ULY northing less than LRY.")
            elif uly == lry:
               raise ValueError ("Grid ULY and LRY are equal, no area to plot")

         else:
            raise ValueError ("Grid projection area could not be checked")

   # print faults
   except ValueError as e:
      exc_type, exc_obj, exc_tb = sys.exc_info()
      fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
      print(exc_type, fname, exc_tb.tb_lineno)
      print(e)
      exit(1)

# ----------------------------------------------------------------------
# SECTION 3.4: Generate the Correct Grid to the desired resolution
#            as implemented 'border', 'global', 'global_all', '25km',
#                     '25km_all', '1km', '100m', '10m', or '1m'.
#            The grid generation is largely handled by other functions

   try:
      # This is the procedure to process grids that are 25km
      # resolution or higher
      if Grid_Res in grid_format[:5]:

         if form_out == "LTM":
            GridX,GridY = generate_LTM_grid(Grid_Res,z,h)

         elif form_out == "LPS":
            GridX,GridY = generate_LPS_grid(Grid_Res,h)

         elif form_out == "LGRS":
            GridX,GridY,multigrid = generate_LGRS_grid(Grid_Res,z,h,
                        LatLon_override=LatLon_override)

         elif form_out == "PolarLGRS":
            GridX,GridY = generate_PolarLGRS_grid(Grid_Res,h,
                     LatLon_override=LatLon_override)

            pass
         else:
            raise ValueError("Operation failed: Large scale grid "
                  "> 25km could not be generated. Check input "
                  "Arguments")

      # plot grids that are < 25km resolution
      elif Grid_Res in grid_format:

         # plot equatorial grid
```

```
        if form_out == "LGRS" or \
          form_out == "LGRS_ACC" or \
          form_out == "ACC":

           GridX,GridY,multigrid = generate_LGRS_grid(Grid_Res,
                                        lonBand=None,
                                        h=None,
                                        z1=z1,
                                        h1=h1,
                                        ulx=ulx,
                                        uly=uly,
                                        z2=z2,
                                        h2=h2,
                                        lrx=lrx,
                                        lry=lry,
                                        multigrid=multigrid,
                                        LatLon_override=LatLon_override)

        # plot polar grid
        elif form_out == "PolarLGRS" or \
            form_out == "PolarLGRS_ACC" or \
            form_out == "PolarACC":

           GridX,GridY = generate_PolarLGRS_grid(Grid_Res,
                                    h=None,
                                    h1=h1,
                                    ulx=ulx,
                                    uly=uly,
                                    h2=h2,
                                    lrx=lrx,
                                    lry=lry,
                                    LatLon_override=LatLon_override)

        else:
           raise ValueError("Grid could not dimension. "
                        "Size could not be determined.")
      else:
        raise ValueError("Operation failed: Grid could not be "
                    "generated.")

     # print faults
    except ValueError as e:
      exc_type, exc_obj, exc_tb = sys.exc_info()
      fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
      print(exc_type, fname, exc_tb.tb_lineno)
      print(e)
      exit(1)

# ----------------------------------------------------------------------
# SECTION 3.5: Convert grid Coordinates to correct output format.
#             The Projections are only LTM, LPS, or LatLon thus the
#             grid coordinates will be in these formats.


    try:
      # The global format is processed in LatLon.
```

```
        # This is the only grid resolution that needs to be converted
        # back to the proper common format. All other grids are
        # generated in LTM or LPS.

        # reconvert global format to LTM or LPS
        if Grid_Res == "global":
            # process equatorial areas
            if ( (form_out == "LTM" ) or \
                 (form_out == "LGRS" ) ):
                  for i in range(0,GridX.shape[0]):
                     for j in range(0,GridX.shape[1]):

                        # break if polygon ends or if no polygon
                        if np.isnan(GridX[i,j]):
                           continue

                        # conversion to LTM
                        zone,hem,GridX[i,j],GridY[i,j] = toLTM(GridX[i,j],GridY[i,j],zone=None,
                           trunc_val=0,process_errors=False)

            # convert polar areas
            elif ( ( form_out == "LPS" ) or \
                   ( form_out == "PolarLGRS" ) ):
                  for i in range(0,GridX.shape[0]):
                     for j in range(0,GridX.shape[1]):

                        # break if polygons ends or if no polygon
                        if np.isnan(GridX[i,j]):
                           continue

                        # Convert to LPS stereographic Coordinates
                        hem,GridX[i,j],GridY[i,j] = toLPS(GridX[i,j],GridY[i,j],
                           trunc_val=0,eqs="Spherical",process_errors=False)

            else:
               raise ValueError ("Output format for grid in global "
                        "format incorrect.")

    except ValueError as e:
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        print(exc_type, fname, exc_tb.tb_lineno)
        print(e)
        exit(1)

# ----------------------------------------------------------------------
# SECTION 3.6: Determine grid cell names in specified output format
    # Determine grid size in meters and the coordinate
    # length. The length is based on a truncated LGRS coordinate,
    # if ACC is used then the coordinate will be shorter and still
    # fit in the coordinate length


    try:
        if attribute:
            # estimated length of output coordinate
```

```
if ( ( Grid_Res == “25km” ) or \
   ( Grid_Res == “25km_all” ) ):
   let_len = 25
elif Grid_Res == “1km”:
   let_len = 9
elif Grid_Res == “100m”:
   let_len = 11
elif Grid_Res == “10m”:
   let_len = 13
elif Grid_Res == “1m”:
   let_len = 15
else:
   let_len = 3

# create binary data for exporting name
GridName = np.chararray((GridX.shape[0],1),itemsize=let_len)

# loop through all grid polygons
for i in range(0,GridX.shape[0]):

   # continue to next polygon if row contained no values
   if np.isnan(GridX[i,0]):
      continue
   else:
      # calculate polygon center to determine grid name
      CentX = np.nanmean(GridX[i,:])
      CentY = np.nanmean(GridY[i,:])

   # at this point the grid center Coordinates are converted
   # to the desired output and written into the output
   # naming array

   # grid name for LTM output
   if form_out == “LTM”: # additional conversion needed
      if Grid_Res == “global_all”:
         z = Calc_LTM_zone(CentX)
         h = Calc_hemisphere(CentY)
      GridName[i,0] = “{}{}”.format(z,h)

   # grid name for LPS output
   elif form_out == “LPS”: # additional conversion needed
      GridName[i,0] = “{}”.format(h)

   # grid name for LGRS grid name
   elif form_out == “LGRS”:
      # to determine names only, the center of grid formatted in
      # LatLon will need to be converted to LTM first
      if multigrid or \
         Grid_Res == “global_all” or \
         Grid_Res == “25km_all” or \
         LatLon_override:

         # convert center of grid location to common format
         z,h,CentX,CentY = toLTM(CentX,CentY,zone=None,
                        trunc_val=0,process_errors=False)
```

```
    # if not multigrid, assign zone and hemisphere to a general
    # term that can be called. Z1 Z2 and H1 and H2 are the
    # same if this is called as multigrid are exported as LatLon
    elif Grid_Res in grid_format[5:]:
        z = z1
        h = h1

    # calculate coordinate for naming
    lonBand,latBand,e25k,n25k,E,N = toLGRS(CentX,CentY,z,h,
                        trunc_val=1,process_errors=True)

    # create names for all resolutions
    if Grid_Res == "global" or Grid_Res == "global_all" :
        GridName[i,0] = "{}{}".format(lonBand,latBand)

    elif Grid_Res == "25km_all" or Grid_Res == "25km":
        GridName[i,0] = "{}{}{}{}".format(lonBand,latBand,e25k,n25k)

    elif Grid_Res == "1km":
        GridName[i,0] = "{}{}{}{}{}{}".format(lonBand,latBand,e25k,n25k,
                                E[:2],N[:2])
    elif Grid_Res == "100m":
        GridName[i,0] = "{}{}{}{}{}{}".format(lonBand,latBand,e25k,n25k,
                                E[:3],N[:3])
    elif Grid_Res == "10m":
        GridName[i,0] = "{}{}{}{}{}{}".format(lonBand,latBand,e25k,n25k,
                                E[:4],N[:4])
    elif Grid_Res == "1m":
        GridName[i,0] = "{}{}{}{}{}{}".format(lonBand,latBand,e25k,n25k,
                                E[:5],N[:5])
    else:
        raise ValueError("Error: LGRS Grid coordinate names could not be "
                         "Generated.")

# process LGRS names for LGRS in ACC format
elif form_out == "LGRS_ACC":
    # an extra conversion is needed for grids generated in LatLon
    if multigrid or \
       Grid_Res == "global_all" or \
       LatLon_override:

        z,h,CentX,CentY = toLTM(CentX,CentY,zone=None,
                        trunc_val=0,process_errors=False)

    elif Grid_Res in grid_format[5:]:
        z = z1
        h = h1

    # calculate coordinate for naming
    lonBand,latBand,e25k,n25k,e1k,E,n1k,N = toLGRS_ACC(CentX,CentY,z,h,
                        trunc_val=1,ACC=False,process_errors=True)

    # write out grid names to the desired grid size
    if Grid_Res == "global" or Grid_Res == "global_all" :
        GridName[i,0] = "{}{}".format(lonBand,latBand)
```

```
        elif Grid_Res == "25km_all" or Grid_Res == "25km":
            GridName[i,0] = "{}{}{}{}".format(lonBand,latBand,e25k,n25k)

        elif Grid_Res == "1km":
            GridName[i,0] = "{}{}{}{}{}{}".format(lonBand,latBand,e25k,n25k,
                                    e1k,n1k)
        elif Grid_Res == "100m":
            GridName[i,0] = "{}{}{}{}{}{}{}{}".format(lonBand,latBand,e25k,n25k,
                                    e1k,E[:1],n1k,N[:1])
        elif Grid_Res == "10m":
            GridName[i,0] = "{}{}{}{}{}{}{}{}".format(lonBand,latBand,e25k,n25k,
                                    e1k,E[:2],n1k,N[:2])
        elif Grid_Res == "1m":
            GridName[i,0] = "{}{}{}{}{}{}{}{}".format(lonBand,latBand,e25k,n25k,
                                    e1k,E,n1k,N)
        else:
            raise ValueError("Error: LGRS ACC Grid coordinate names could not be "
                        "Generated.")

    # For grids <25km resolution, write out names in ACC format
    # truncated to 10m without global location reference
    elif form_out == "ACC":
        # convert LatLon formatted grids back to common
        # format before naming
        if multigrid or LatLon_override:
            z,h,CentX,CentY=toLTM(CentX,CentY,zone=None,
                        trunc_val=0,process_errors=False)

        # Set projection zones, this shouldn't be called by any
        # zone crossing in multiple zones as they are formatted
        # in LatLon to ensure naming. Only grids in one
        # zone area up here
        elif Grid_Res in grid_format[5:]:
            z = z1
            h = h1

        # convert coordinates for LGRS ACC format
        e1k,E,n1k,N = toLGRS_ACC(CentX,CentY,z,h,
                        trunc_val=10,ACC=True,process_errors=True)

        if Grid_Res == "1km":
            GridName[i,0] = "{}{}".format(e1k,n1k)
        elif Grid_Res == "100m":
            GridName[i,0] = "{}{}{}{}".format(e1k,E[:1],n1k,N[:1])
        elif Grid_Res == "10m":
            GridName[i,0] = "{}{}{}{}".format(e1k,E[:2],n1k,N[:2])
        elif Grid_Res == "1m":
            raise ValueError("Error: ACC Grid coordinate cannot be a 1m resolution. Plot "
                        "LGRS_ACC grid instead.")
        elif Grid_Res in grid_format:
            raise ValueError("Error: ACC Grid coordinate specified incorrectly."
                        "Check input grid resolution")
        else:
            raise ValueError("Error: ACC Grid coordinate names could not be "
                        "generated.")
```

```
# names for polar LGRS
elif form_out == “PolarLGRS”:
   # extra conversion step for grids formatted in LatLon
   if multigrid or \
     Grid_Res == “global_all” or \
     LatLon_override:

      # LatLon to LPS
      h,CentX,CentY = toLPS(CentX,CentY,trunc_val=0,
                  eqs=”Spherical”,process_errors=True)

   # grids formatted within a box need a common variable for
   # output. This has no effect as grids within only one zone
   # make it to this check
   elif Grid_Res in grid_format[5:]:
      h = h1

   # convert LPS coordinates to determine name
   lonBand,e25k,n25k,E,N = toLGRS_polar(CentX,CentY,h,
                  trunc_val=1,process_errors=True)

   # format name for output Coordinates
   if Grid_Res == “global” or Grid_Res == “global_all” :
      GridName[i,0] = “{}”.format(lonBand)

   elif Grid_Res == “25km_all” or Grid_Res == “25km”:
      GridName[i,0] = “{}{}{}”.format(lonBand,e25k,n25k)

   elif Grid_Res == “1km”:
      GridName[i,0] = “{}{}{}{}{}”.format(lonBand,e25k,n25k,
                        E[:2],N[:2])
   elif Grid_Res == “100m”:
      GridName[i,0] = “{}{}{}{}{}”.format(lonBand,e25k,n25k,
                        E[:3],N[:3])
   elif Grid_Res == “10m”:
      GridName[i,0]=”{}{}{}{}{}”.format(lonBand,e25k,n25k,
                        E[:4],N[:4])
   elif Grid_Res == “1m”:
      GridName[i,0] = “{}{}{}{}{}”.format(lonBand,e25k,n25k,
                        E[:5],N[:5])
   else:
      raise ValueError(“Error: PolarLGRS Grid coordinate names could not be “
            “generated.”)

# Polar LGRS coordinate names in ACC format without truncation
elif form_out == “PolarLGRS_ACC”:

   # extra processing step for grids formatted in LatLon
   if multigrid or \
     Grid_Res == “global_all” or \
     LatLon_override:

      # convert to LPS
      h, CentX, CentY = toLPS(CentX, CentY,
         trunc_val=0,
         eqs=”Spherical”,
```

```
            process_errors=False)


        # assign projection hemisphere to common term for exporting
        elif Grid_Res in grid_format[5:]:
            h=h1

        # Convert center of grid area to determine name
        lonBand,e25k,n25k,e1k,E,n1k,N = toLGRS_polar_ACC(CentX,CentY,h,
                        trunc_val=1,ACC=False,process_errors=True)

        # format output names for all grid resolutions
        if Grid_Res == "global" or Grid_Res == "global_all" :
            GridName[i,0] = "{}".format(lonBand)

        elif Grid_Res == "25km_all" or Grid_Res == "25km":
            GridName[i,0] = "{}{}{}".format(lonBand,e25k,n25k)

        elif Grid_Res == "1km":
            GridName[i,0] = "{}{}{}{}{}{}".format(lonBand,e25k,n25k,
                                e1k,n1k)
        elif Grid_Res == "100m":
            GridName[i,0] = "{}{}{}{}{}{}{}{}".format(lonBand,e25k,n25k,
                                e1k,E[:1],n1k,N[:1])
        elif Grid_Res == "10m":
            GridName[i,0] = "{}{}{}{}{}{}{}{}".format(lonBand,e25k,n25k,
                                e1k,E[:2],n1k,N[:2])
        elif Grid_Res == "1m":
            GridName[i,0] = "{}{}{}{}{}{}{}{}".format(lonBand,e25k,n25k,
                                e1k,E,n1k,N)
        else:
            raise ValueError("Error: PolarLGRS ACC Grid coordinate names could not be "
                        "Generated.")

    # Polar LGRS coordinate names in ACC format with truncation
    elif form_out == "PolarACC":

        # grids covering multiple zones are in LatLon and need to
        # be converted to LTM before names can be determined
        if multigrid or LatLon_override:

            # convert to LTM
            h, CentX, CentY = toLPS(CentX, CentY,
                trunc_val=0,
                eqs="Spherical",
                process_errors=False)


        # For grids in single zone assign output hemisphere
        elif Grid_Res in grid_format[5:]:
            h = h1

        # determine PolarACC Grid Names
        e1k,E,n1k,N = toLGRS_polar_ACC(CentX,CentY,h,
                    trunc_val=10,ACC=True,process_errors=True)
```

```
            # format output names
            if Grid_Res == "1km":
               GridName[i,0] = "{}{}".format(e1k,n1k)
            elif Grid_Res == "100m":
               GridName[i,0] = "{}{}{}{}".format(e1k,E[:1],n1k,N[:1])
            elif Grid_Res == "10m":
               GridName[i,0] = "{}{}{}{}".format(e1k,E[:2],n1k,N[:2])
            elif Grid_Res == "1m":
               raise ValueError("Error: Polar ACC Grid coordinate is 1m resolution. Plot "
                        "LGRS_ACC grid instead.")
            elif Grid_Res in grid_format:
               raise ValueError("Error: PolarACC Grid coordinate specified incorrectly."
                        "Check input grid resolution")
            else:
               raise ValueError("Error: ACC Grid coordinate names could not be "
                        "generated.")
      else:
         GridName = None

   # report errors if discovered
   except ValueError as e:
      exc_type, exc_obj, exc_tb = sys.exc_info()
      fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
      print(exc_type, fname, exc_tb.tb_lineno)
      print(e)
      exit(1)

# ----------------------------------------------------------------------
# SECTION 3.7: Assign output WKT1 Projection string for writing file


   try:
      # Export in LatLon Coordinates
      if ( ( multigrid ) or
           ( LatLon_override ) or \
           ( Grid_Res == "border" ) or \
           ( Grid_Res == "global_all" ) or \
           ( Grid_Res == "25km_all" ) ):
         WKT1 = Display_geocentric_degrees

      # export Equatorial Grids
      else:
         if ( ( form_out == "LTM" ) or \
            ( form_out == "LGRS" ) or \
            ( form_out == "LGRS_ACC" ) or \
            ( form_out == "ACC" ) ):

            # Step used to assign correct zone
            if Grid_Res in grid_format[5:]:
               z = z1
               h = h1

            # determine LTM zone central meridian
            lam0 = Calc_LTM_prime_meridian(z,"deg")

            # Depending on hemisphere assign correct projection
```

```
            # to data
            if h == "N":
               WKT1 = N_LTM_projection_meters.format(lam0)
            elif h == "S":
               WKT1 = S_LTM_projection_meters.format(lam0)

         # export polar grids
         elif ( ( form_out == "LPS" )  or \
            ( form_out == "PolarLGRS" ) or \
            ( form_out == "PolarLGRS_ACC" ) or \
            ( form_out == "PolarACC" ) ):

            # step used to determine hemisphere
            if Grid_Res in grid_format[5:]:
               h = h1

            # assign projection based on the hemispheric
            # location
            if h == "N":
               WKT1 = N_LPS_projection_meters
            elif h == "S":
               WKT1 = S_LPS_projection_meters
         else:
            raise ValueError ("Map output projection could not be determined")

   except ValueError as e:
      exc_type, exc_obj, exc_tb = sys.exc_info()
      fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
      print(exc_type, fname, exc_tb.tb_lineno)
      print(e)
      exit(1)

# ----------------------------------------------------------------------
# SECTION 3.8:  Write grid to Esri shapefile and format output names


   try:
      # This is the procedure to process grids that are 25km
      # resolution or higher

      # format shapefile output names for equatorial region
      if ( ( form_out == "LTM" ) or \
          ( form_out == "LGRS" ) or \
          ( form_out == "LGRS" ) or \
          ( form_out == "LGRS_ACC" ) or \
          ( form_out == "ACC" ) ):

         # global grid names
         if Grid_Res == "global_all" or Grid_Res == "25km_all":
            f_o = "{}_{}_Grid".format(form_out,Grid_Res)
         else:
            # multigrid names for grid passing through multiple zones
            if multigrid:
               f_o = "{}_{}_{}{}_MultiGrid".format(form_out,Grid_Res,z,h)

            # Output name for grids in LTM equatorial regions
```

```
            else:
                f_o = "{}_{}_{}{}_Grid".format(form_out,Grid_Res,z,h)

        # output names for polar regions
        else:
            f_o = "{}_{}_{}_Grid".format(form_out,Grid_Res,h)

        # write out to shapefile for single polygon
        if Grid_Res == "border" or Grid_Res == "global":

            # Export LTM and LPS single polygons
            if form_out == "LTM" or form_out == "LPS":
                Xport_polygon(GridX,
                              GridY,
                              GridName,
                              f_o,
                              WKT1,
                              attribute,
                              output_file_form,
                              overwrite)

            # only LTM and LPS have single polygon grids
            # all other global grids are processed in
            # multipolygon geometries
            else:
                Xport_multipolygons(GridX,
                                    GridY,
                                    GridName,
                                    f_o,
                                    WKT1,
                                    attribute,
                                    output_file_form,
                                    overwrite)

        # export all other grids as a multipolygon
        elif Grid_Res in grid_format:
            Xport_multipolygons(GridX,
                                GridY,
                                GridName,
                                f_o,
                                WKT1,
                                attribute,
                                output_file_form,
                                overwrite)
        else:
            raise ValueError ("Grid could not be exported. "
                              "Please check function input values ")

        # print faults
    except ValueError as e:
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        print(exc_type, fname, exc_tb.tb_lineno)
        print(e)
        exit(1)
```

```
# ---------------------------------------------------------------------
# Print program timing and elapsed time


    if info:
        EndTime = datetime.datetime.now()
        print('\n', ProgName, 'execution time')
        print('    Start Time:', StartTime)
        print('    End Time:  ', EndTime)
        print('    Run Time:  ', EndTime - StartTime)


# =====================================================================
# Run main program
# determine input argument format

# main program control and operations
multigrid = False  # process grid across LTM area
attribute = True   # write out shapefile attribute information
overwrite = True   # delete and rewrite file if it exits
output_file_form = "shp"  # set the type output file.
                    # shp for Esri shapefile
info = True  # print out program information and processing time
LatLon_override = False # force the grids to output in LatLon
                    # only applies to LGRS and LGRS in ACC format
                    # NOTE: these projections must still be used
                    # in LPS or LTM to ensure correct spatial
                    # dimensions.

# initialize program information
if info:
    StartTime = datetime.datetime.now()
    print('Program:', ProgName)


try:
    # read in initial arguments
    initialize_LGRS_grid_generation_globals()
    form_in = sys.argv[1]
    form_out = sys.argv[2]
    grid_res = sys.argv[3]

# raise problem if no basic information is provided to the
# program
except IndexError:
    form_in = None
    form_out = None
    grid_res = None
    raise ValueError("Input, output, or grid resolution specified "
        "incorrectly.\n Select one of the following:\n"
        "I/O:\n{}\n\nGrid Res:\n{}".format(coordinate_format_str,
                                    grid_format_str))

# run the grid generation program if enough arguments are provided
try:
    if ( (form_in in coordinate_format) and \
```

```
        ( (form_out in coordinate_format) or \
          (form_out == “ACC”) or \
          (form_out == “PolarACC”) ) and \
        ( grid_res in grid_format ) ):

        # call main function to generate grids
        main(form_in,form_out,grid_res,
                        multigrid,
                        attribute,
                        overwrite,
                        output_file_form,
                        info,
                        LatLon_override)

    else:
        raise ValueError(“Input, output, or grid resolution specified “
              “incorrectly.\n Select one of the following:\n”
              “I/O:\n{}\n\nGrid Res:\n{}”.format(coordinate_format_str,
                                    grid_format_str))

except ValueError as e:

    exc_type, exc_obj, exc_tb = sys.exc_info()
    fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
    print(exc_type, fname, exc_tb.tb_lineno)
    print(e)

# ============================================================================
```

Appendix 3. Lunar Transverse Mercator Map Projection Well-Known Text

This appendix contains complete well-known text (WKT) for all 90 Lunar Transverse Mercator (LTM) zones using the transverse Mercator map projection. WKT is a common geographic information system (GIS) format for projecting geospatial information as well as spatial data for planetary science (Hare and Malapert, 2021). As written, the WKT below can be copied into most GIS packages or into geospatial projection sidecar files. The WKT for Lunar Polar Stereographic (LPS) zones is provided in the "LPS Map Projection Zones and Sample WKT" section.

Zone LTM_1N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-176,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_1N.”],
    BBOX[82.,-180,0.,-172]]]
```

Zone LTM_1S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-176,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_1S.”],
    BBOX[0.,-180,-82.,-172]]]
```

Zone LTM_2N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-168,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_2N.”],
    BBOX[82.,-172,0.,-164]]]
```

Zone LTM_2S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-168,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_2S.”],
    BBOX[0.,-172,-82.,-164]]]
```

Zone LTM_3N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-160,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_3N.”],
     BBOX[82.,-164,0.,-156]]]
```

Zone LTM_3S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-160,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_3S.”],
    BBOX[0.,-164,-82.,-156]]]
```

Zone LTM_4N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-152,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_4N.”],
    BBOX[82.,-156,0.,-148]]]
```

Zone LTM_4S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-152,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_4S.”],
    BBOX[0.,-156,-82.,-148]]]
```

Zone LTM_5N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-144,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_5N.”],
    BBOX[82.,-148,0.,-140]]]
```

Zone LTM_5S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-144,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_5S.”],
    BBOX[0.,-148,-82.,-140]]]
```

Zone LTM_6N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-136,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_6N.”],
    BBOX[82.,-140,0.,-132]]]
```

Zone LTM_6S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-136,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_6S.”],
    BBOX[0.,-140,-82.,-132]]]
```

Zone LTM_7N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-128,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_7N.”],
    BBOX[82.,-132,0.,-124]]]
```

Zone LTM_7S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-128,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_7S.”],
    BBOX[0.,-132,-82.,-124]]]
```

Zone LTM_8N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-120,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_8N.”],
    BBOX[82.,-124,0.,-116]]]
```

Zone LTM_8S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-120,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_8S.”],
    BBOX[0.,-124,-82.,-116]]]
```

Zone LTM_9N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-112,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_9N.”],
    BBOX[82.,-116,0.,-108]]]
```

Zone LTM_9S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-112,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_9S.”],
    BBOX[0.,-116,-82.,-108]]]
```

Zone LTM_10N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-104,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_10N.”],
    BBOX[82.,-108,0.,-100]]]
```

Zone LTM_10S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-104,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_10S.”],
    BBOX[0.,-108,-82.,-100]]]
```

Zone LTM_11N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-96,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_11N.”],
    BBOX[82.,-100,0.,-92]]]
```

Zone LTM_11S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-96,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_11S.”],
    BBOX[0.,-100,-82.,-92]]]
```

Zone LTM_12N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-88,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_12N.”],
    BBOX[82.,-92,0.,-84]]]
```

Zone LTM_12S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-88,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_12S.”],
    BBOX[0.,-92,-82.,-84]]]
```

Zone LTM_13N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-80,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_13N.”],
    BBOX[82.,-84,0.,-76]]]
```

Zone LTM_13S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-80,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_13S.”],
    BBOX[0.,-84,-82.,-76]]]
```

Zone LTM_14N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-72,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_14N.”],
    BBOX[82.,-76,0.,-68]]]
```

Zone LTM_14S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-72,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_14S.”],
    BBOX[0.,-76,-82.,-68]]]
```

Zone LTM_15N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-64,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_15N.”],
    BBOX[82.,-68,0.,-60]]]
```

Zone LTM_15S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-64,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_15S.”],
    BBOX[0.,-68,-82.,-60]]]
```

Zone LTM_16N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-56,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_16N.”],
    BBOX[82.,-60,0.,-52]]]
```

Zone LTM_16S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-56,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_16S.”],
    BBOX[0.,-60,-82.,-52]]]
```

Zone LTM_17N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-48,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_17N.”],
    BBOX[82.,-52,0.,-44]]]
```

Zone LTM_17S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-48,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_17S.”],
    BBOX[0.,-52,-82.,-44]]]
```

Zone LTM_18N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-40,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_18N.”],
    BBOX[82.,-44,0.,-36]]]
```

Zone LTM_18S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-40,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_18S.”],
    BBOX[0.,-44,-82.,-36]]]
```

Zone LTM_19N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-32,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_19N.”],
    BBOX[82.,-36,0.,-28]]]
```

Zone LTM_19S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-32,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_19S.”],
    BBOX[0.,-36,-82.,-28]]]
```

Zone LTM_20N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-24,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_20N.”],
    BBOX[82.,-28,0.,-20]]]
```

Zone LTM_20S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-24,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_20S.”],
    BBOX[0.,-28,-82.,-20]]]
```

Zone LTM_21N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-16,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_21N.”],
    BBOX[82.,-20,0.,-12]]]
```

Zone LTM_21S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-16,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_21S.”],
    BBOX[0.,-20,-82.,-12]]]
```

Zone LTM_22N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-8,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_22N.”],
    BBOX[82.,-12,0.,-4]]]
```

Zone LTM_22S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,-8,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_22S.”],
    BBOX[0.,-12,-82.,-4]]]
```

Zone LTM_23N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_23N.”],
    BBOX[82.,-4,0.,4]]]
```

Zone LTM_23S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_23S.”],
    BBOX[0.,-4,-82.,4]]]
```

Zone LTM_24N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,8,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_24N.”],
    BBOX[82.,4,0.,12]]]
```

Zone LTM_24S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,8,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_24S.”],
    BBOX[0.,4,-82.,12]]]
```

Zone LTM_25N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,16,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_25N.”],
    BBOX[82.,12,0.,20]]]
```

Zone LTM_25S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,16,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_25S.”],
    BBOX[0.,12,-82.,20]]]
```

Zone LTM_26N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,24,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_26N.”],
    BBOX[82.,20,0.,28]]]
```

Zone LTM_26S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,24,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_26S.”],
    BBOX[0.,20,-82.,28]]]
```

Zone LTM_27N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,32,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_27N.”],
    BBOX[82.,28,0.,36]]]
```

Zone LTM_27S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,32,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_27S.”],
    BBOX[0.,28,-82.,36]]]
```

Zone LTM_28N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,40,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_28N.”],
    BBOX[82.,36,0.,44]]]
```

Zone LTM_28S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,40,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_28S.”],
    BBOX[0.,36,-82.,44]]]
```

Zone LTM_29N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,48,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_29N.”],
    BBOX[82.,44,0.,52]]]
```

Zone LTM_29S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,48,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_29S.”],
    BBOX[0.,44,-82.,52]]]
```

Zone LTM_30N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,56,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_30N.”],
    BBOX[82.,52,0.,60]]]
```

Zone LTM_30S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,56,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_30S.”],
    BBOX[0.,52,-82.,60]]]
```

Zone LTM_31N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,64,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_31N.”],
    BBOX[82.,60,0.,68]]]
```

Zone LTM_31S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,64,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_31S.”],
    BBOX[0.,60,-82.,68]]]
```

Zone LTM_32N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,72,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_32N.”],
    BBOX[82.,68,0.,76]]]
```

Zone LTM_32S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,72,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_32S.”],
    BBOX[0.,68,-82.,76]]]
```

Zone LTM_33N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,80,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_33N.”],
    BBOX[82.,76,0.,84]]]
```

Zone LTM_33S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,80,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_33S.”],
    BBOX[0.,76,-82.,84]]]
```

Zone LTM_34N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,88,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_34N.”],
    BBOX[82.,84,0.,92]]]
```

Zone LTM_34S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,88,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_34S.”],
    BBOX[0.,84,-82.,92]]]
```

Zone LTM_35N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,96,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_35N.”],
    BBOX[82.,92,0.,100]]]
```

Zone LTM_35S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,96,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_35S.”],
    BBOX[0.,92,-82.,100]]]
```

Zone LTM_36N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,104,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_36N.”],
    BBOX[82.,100,0.,108]]]
```

Zone LTM_36S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,104,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_36S.”],
    BBOX[0.,100,-82.,108]]]
```

Zone LTM_37N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,112,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_37N.”],
    BBOX[82.,108,0.,116]]]
```

Zone LTM_37S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,112,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_37S.”],
    BBOX[0.,108,-82.,116]]]
```

Zone LTM_38N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,120,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_38N.”],
    BBOX[82.,116,0.,124]]]
```

Zone LTM_38S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,120,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_38S.”],
    BBOX[0.,116,-82.,124]]]
```

Zone LTM_39N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,128,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_39N.”],
    BBOX[82.,124,0.,132]]]
```

Zone LTM_39S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,128,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_39S.”],
    BBOX[0.,124,-82.,132]]]
```

Zone LTM_40N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,136,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_40N.”],
    BBOX[82.,132,0.,140]]]
```

Zone LTM_40S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,136,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_40S.”],
     BBOX[0.,132,-82.,140]]]
```

Zone LTM_41N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,144,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_41N.”],
    BBOX[82.,140,0.,148]]]
```

Zone LTM_41S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,144,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_41S.”],
    BBOX[0.,140,-82.,148]]]
```

Zone LTM_42N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,152,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_42N.”],
    BBOX[82.,148,0.,156]]]
```

Zone LTM_42S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,152,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_42S.”],
    BBOX[0.,148,-82.,156]]]
```

Zone LTM_43N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,160,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_43N.”],
    BBOX[82.,156,0.,164]]]
```

Zone LTM_43S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,160,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_43S.”],
    BBOX[0.,156,-82.,164]]]
```

Zone LTM_44N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,168,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_44N.”],
    BBOX[82.,164,0.,172]]]
```

Zone LTM_44S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,168,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_44S.”],
    BBOX[0.,164,-82.,172]]]
```

Zone LTM_45N

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,176,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,0,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_45N.”],
    BBOX[82.,172,0.,180]]]
```

Zone LTM_45S

```
PROJCRS[“Moon (2015) - Sphere / Ocentric / Tranverse Mercator”,
  BASEGEOGCRS[“Moon (2015) - Sphere / Ocentric”,
    DATUM[“Moon (2015) - Sphere”,
      ELLIPSOID[“Moon (2015) - Sphere”,1737400,0,
        LENGTHUNIT[“metre”,1]]],
    PRIMEM[“Reference Meridian”,0,
      ANGLEUNIT[“degree”,0.0174532925199433]],
    ID[“IAU”,30100,2015]],
  CONVERSION[“transverse Mercator”,
    METHOD[“transverse Mercator”,
      ID[“EPSG”,9807]],
    PARAMETER[“Latitude of natural origin”,0,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8801]],
    PARAMETER[“Longitude of natural origin”,176,
      ANGLEUNIT[“degree”,0.0174532925199433],
      ID[“EPSG”,8802]],
    PARAMETER[“Scale factor at natural origin”,.999,
      SCALEUNIT[“unity”,1],
      ID[“EPSG”,8805]],
    PARAMETER[“False easting”,250000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8806]],
    PARAMETER[“False northing”,2500000,
      LENGTHUNIT[“metre”,1],
      ID[“EPSG”,8807]]],
  CS[Cartesian,2],
    AXIS[“(E)”,east,
      ORDER[1],
      LENGTHUNIT[“metre”,1]],
    AXIS[“(N)”,north,
      ORDER[2],
      LENGTHUNIT[“metre”,1]],
  USAGE[
    SCOPE[“Lunar Transverse Mercator Zone LTM_45S.”],
    BBOX[0.,172,-82.,180]]]
```

Moffett Field Publishing Service Center
Manuscript approved December 20, 2024
Edited by Regan Austin
Illustration support by Katie Sullivan
Layout by Cory Hurd

ISSN 2328-7055 (online)
https://doi.org/10.3133/tm11E1