Part 650 – Engineering Field Handbook

Chapter 19 – Hydrology Tools for Wetland Identification and Analysis

Prairie Pothole Region

States of Iowa, Minnesota, North Dakota, and South Dakota

PPR650.1900 Introduction

The purpose of this supplement is to document the procedures that were utilized to generate lateral effect distance recommendations in the Prairie Pothole Region (PPR) States of Iowa, Minnesota, North Dakota, and South Dakota. A report was developed using structured query language (SQL) to query and format data from the National Soil Information System (NASIS). The report results for soils in the PPR are made available to the public on the Web Soil Survey. The SQL report is designed to assign a hydrogeomorphic (HGM) classification, and subclass if appropriate, to hydric soil components and the assigned classification is then used to calculate lateral effect. The report was run on the official soil survey data as of November 2021, and the resulting lateral effect calculations database is intended to be static. Any future changes will occur only when the four State Conservationists agree results should be refreshed to reflect updates to the official Soil Survey.

PPR650.1903 Assigning the Hydrogeomorphic Wetland Classes to PPR soils

A. Introduction to Hydrogeomorphic Wetland Classification

HGM wetland classes are based on landscape position, dominant water source, and hydrodynamics (direction and hydraulic head of water movement). The foundations of HGM wetland classification are provided in Brinson (1993) and Smith et al. (1995). Technical Note 190-8-76, "Hydrogeomorphic Wetland Classification" (TN-190-8-76) slightly modified the original concepts to ensure consistent classification by NRCS. There are seven HGM wetland classes defined in TN-190-8-76 and identified in Title 210, National Engineering Handbook, Part 650, Chapter 19 "Hydrology Tools for Wetland Identification" (210-NEH-650.19). There are three subclasses of the DEPRESSIONAL class (210-NEH-650.19) and one subclass of SLOPE that are important for drainage equation application that are also listed below.

(i) RIVERINE
(ii) ESTUARINE FRINGE
(iii) LACUSTRINE FRINGE
(iv) SLOPE
- Fen

(v) ORGANIC SOIL FLATS
(vi) MINERAL SOIL FLATS
(vii) DEPRESSIONAL
- Recharge
- Discharge and Flow through

C. Soil Survey Definitions

(6) The following official soil survey definitions and criteria are important in assigning HGM wetland classes and subclasses:

(i) Flooding: The annual probability of a flood event (inundation by flowing water), expressed as a class. (SSM).

- Where the flooding frequency is:
  - Very frequent: Flooding is likely to occur very often under usual weather conditions; more than 50 percent chance in all months of any year;
  - Frequent: Flooding is likely to occur often under usual weather conditions; more than 50 percent chance of flooding in any year or more than 50 times in 100 years, but less than a 50 percent chance of flooding in all months in any year;
  - Occasional: Flooding is expected infrequently, 5 to 50 percent chance in any year, (5 to 50 times in 100 years);
  - Rare: Flooding is unlikely but possible under unusual weather conditions; 1 to 5 percent chance in any year (1 to 5 times in 100 years);
  - Very rare: Flooding is very unlikely but is possible under unusual weather conditions; less than 1 percent chance in any year (less than 1 time in 100 years, but more than 1 time in 500 years); or
  - None: No reasonable possibility of flooding; near 0 percent chance of flooding in any year or less than 1 time in 500 years; and
- Where the flooding duration is the average duration of inundation per flood occurrence expressed as a class is:
  - Very long: More than 30 days;
  - Long: 7 days to 30 days;
  - Brief: 2 days to 7 days;
  - Very brief: 4 hours to 48 hours; or
  - Extremely brief: 0.1 to 4 hours; and

- Where landform position (components on flooded landforms). Soils with a very rare flooding frequency are often not on the hydric soil list. Landforms that are subject to flooding include bars, channels, drainageways, flood plains, flood-plain steps, and oxbows.

(ii) Ponding: The annual probability of a ponding event (inundation by standing water), expressed as a class.

- Where the ponding frequency is:
  - Frequent: Ponding is likely to occur under usual weather conditions; more than 50 percent chance in any year, or more than 50 times in 100 years;
  - Occasional: Ponding is expected infrequently under usual weather conditions; 5 to 50 percent chance of ponding in any year, or 5 to 50 times in 100 years;
  - Rare: Ponding unlikely but possible under unusual weather conditions; from nearly 0 to 5 percent chance of ponding in any year, or nearly 0 to 5 times in 100 years; or
  - None: No reasonable possibility of ponding, near 0 percent chance on ponding in any year; and
- Where ponding duration is the average duration, or length of time, of the ponding occurrence expressed as a class is:
  - Very long: More than 30 days;
  - Long: 7 days to 30 days;
  - Brief: 2 days to 7 days; or
  - Very brief: 4 hours to 48 hours.

(iii) Histic Epipedon: The histic epipedon is a layer (one or more horizons) that is characterized by saturation (for 30 days or more, cumulative) and reduction for some time during normal years (or is artificially drained) and either:

- Consists of organic soil material that:
  - Is 20 to 60 cm thick and either contains 75 percent or more (by volume) Sphagnum fibers or has a bulk density, moist, of less than 0.1 g/cm3; or
  - Is 20 to 40 cm thick; or
  - Is an Ap horizon that, when mixed to a depth of 25 cm, has an organic-carbon content (by weight) of:
    - 16 percent or more if the mineral fraction contains 60 percent or more clay; or
    - 8 percent or more if the mineral fraction contains no clay; or
    - 8 + (clay percentage divided by 7.5) percent or more if the mineral fraction contains less than 60 percent clay.
- Most histic epipedons consist of organic soil material (defined in Chapter 2: Soil Survey Staff, 2014). Item ii provides for a histic epipedon that is an Ap horizon consisting of mineral soil material. A histic epipedon consisting of mineral soil material can also be part of a mollic or umbric epipedon.

(iv) Discharge or Recharge: Presence of secondary soluble salts within 50 cm of the soil surface is used to indicate the presence of discharge conditions. A calcium carbonate equivalence greater than or equal to 1 percent; a gypsum content greater than or equal to 1 percent; or an electrical conductivity greater than 4 ds/m is considered evidence that the hydric soil functions in a discharge capacity. Likewise, the absence of said secondary salts within 50 cm of the soil surface is used to indicate that the soil functions in a recharge capacity. Soils have an inherent, net negative charge obtained from the clay sized particles. Vegetation that is not salt tolerant exhibits osmotic stress from the presence of salts when the electrical conductivity is greater than 4 ds/m.

(v) Sodicity: Soils affected by sodium salts are separated from those affected by other soluble salts because sodium dispersion alters the soil hydrology. This dispersion negatively affects the water infiltration. Sodium is known to affect soil physical properties when the sodium adsorption ratio is 5 or greater.

(7) Hydric Soils List Criteria: Wetland conditions require the presence of hydric soils. The USDA-NRCS Soil and Plant Sciences Division uses the following criteria to designate soil components as hydric in the NASIS database. Soil components must meet one or more of these criteria to be considered a member of a HGM wetland class. Any soils meeting one or more of the criteria below are considered to be a member of an HGM wetland class:

(i) All Histels except Folistels and Histosols except Folists; or

(ii) Map unit components in Aquic suborders, great groups, or subgroups, Albolls suborder, Historthels great group, Histoturbels great group, or Andic, Cumulic, Pachic, or Vitrandic subgroups that:

- Based on the range of characteristics for the soil series, will at least in part meet one or more Field Indicators of Hydric Soils in the United States, or
- Show evidence that the soil meets the definition of a hydric soil;

(iii) Map unit components that are frequently ponded for long duration or very long duration during the growing season that:

- Based on the range of characteristics for the soil series, will at least in part meet one or more Field Indicators of Hydric Soils in the United States or
- Show evidence that the soil meets the definition of a hydric soil; or

(iv) Map unit components that are frequently flooded for long duration or very long duration during the growing season that:

- Based on the range of characteristics for the soil series, will at least in part, meet one or more Field Indicators of Hydric Soils in the United States , or
- Show evidence that the soils meet the definition of a hydric soil.

E. Hierarchical Approach to Assigning HGM Class

(1) The script used soil properties and qualities to differentiate and assign HGM wetland classes and subclasses including hydric rating, flooding frequency, presence of soluble salts (electrical conductivity, sodium adsorption ratio, calcium carbonate equivalent, and gypsum), presence of a histic epipedon, and landform. These properties and qualities are described in-depth in section PPR650.1903C. Only soil components that have a hydric rating of "yes" were assigned an HGM wetland class (see section PPR650.190C(7) for hydric screening criteria used for the procedure). Soil components on the hydric soil list met one or more of the criteria documented in Federal Register Document 2012-4733, filed February 29, 2012.

(2) The script assigned wetland classes in a hierarchical manner, which is listed in numbered bullets below. Wetland classes ESTUARINE FRINGE, LACUSTRINE FRINGE, and Flow through DEPRESSIONAL were not included in this approach because there is not enough information on soil properties or qualities in the National Soil Information System (NASIS) to make these classifications using offsite methods. Soil properties and qualities do not identify proximity to cyclic action of ocean tides, lake levels, or precisely equal groundwater inflow compared to outflow. Furthermore, there are no oceans near the PPR states, which excludes ESTUARINE FRINGE wetland from consideration. LACUSTRINE FRINGE wetlands should be identified on a case-by-case basis when lake levels influence wetland hydrology. Flow through DEPRESSIONAL require equal groundwater inflow to groundwater outflow and can include a combination of recharge and discharge characteristics. Flow through DEPRESSIONAL wetland classes cannot be ascertained solely using the official soil survey data; instead, assignments differentiate as either recharge DEPRESSIONAL or discharge DEPRESSIONAL.

(3) Assignment started at the lowest part of the landscape and moved to the highest part. This process ensured that once a component was assigned a wetland class, the component was not considered a member of any other wetland class. The exception to this hierarchical approach is the fen SLOPE, which are unique. They are typically higher on the landscape and have a histic epipedon that results from very nearly permanent saturation. A summary of assignments is listed below, and the NASIS script utilized is included in section PPR650.1916A.

(i) Members of the fen SLOPE subclass were distinguished by the presence of a histic epipedon greater than or equal to 20 cm thick and were assigned the landform "fen."

(ii) Members of the ORGANIC SOILS FLAT wetland class are not a member of a previously defined class and were distinguished by the presence of a histic epipedon greater than or equal to 20 cm thick.

(iii) A component was considered a member of the RIVERINE wetland class if it was not a member of a previously defined class and the flooding frequency was very frequent, frequent, occasional, or rare; it is on a flooded landform; a histic epipedon was not present; and the sodium adsorption ratio was less than 5 within 50 cm of the soil surface. Landforms that are subject to flooding include channels, drainageways, flood plains, flood plain steps, and oxbows.

(iv) A component was considered a member of the RIVERINE, sodic, wetland class and subclass if it was not a member of a previously defined class and the flooding frequency was very frequent, frequent, occasional, or rare; it was on a flooded landform; a histic epipedon was not present; and the sodium adsorption ratio was greater than or equal to 5 within 50 cm of the soil surface. Landforms that are subject to flooding include channels, drainageways, flood plains, flood plain steps, and oxbows.

(v) A component was considered a member of the MINERAL SOILS FLAT wetland class if it was not a member of a previously defined class, a histic epipedon was not present, the sodium adsorption ratio was less than 5 within 50 cm of the soil surface, the component was not subject to flooding, and the landform character was typically broad and flat.

(vi) A component was considered a member of the MINERAL SOILS FLAT, sodic wetland class and subclass if it was not a member of a previously defined class, a histic epipedon was not present, the sodium adsorption ratio was greater than or equal to 5 within 50 cm of the soil surface, the component was not subject to flooding, and the landform character was typically broad and flat.

(vii) Members of the discharge DEPRESSIONAL are not a member of a previously defined class, do not have a histic epipedon, have secondary salts within 50 cm of the soil surface, and have a sodium adsorption ratio of less than 5 within 50 cm of the soil surface.

(viii) Members of the sodic discharge DEPRESSIONAL are not a member of a previously defined class, do not have a histic epipedon, have secondary salts within 50 cm of the soil surface, and have a sodium adsorption ratio of 5 or greater within 50 cm of the soil surface.

(ix) Members of the SLOPE wetland class are not a member of a previously defined class, do not have a histic epipedon, have a sodium adsorption ratio of less than 5 within 50 cm of the soil surface, and are not subject to flooding.

(x) Members of the SLOPE, sodic wetland class are not a member of a previously defined class, do not have a histic epipedon, have a sodium adsorption ratio of greater than or equal to 5 within 50 cm of the soil surface, and are not subject to flooding.

(xi) Members of the recharge DEPRESSIONAL are not a member of a previously defined class, do not have a histic epipedon, do not have secondary salts within 50 cm of the soil surface, and have a sodium adsorption ratio of less than 5 within 50 cm of the soil surface.

(xii) Hydric soil components on geomorphic positions that are anthropogenic (affected by people) were not assigned a wetland class and were considered ARTIFICIAL.

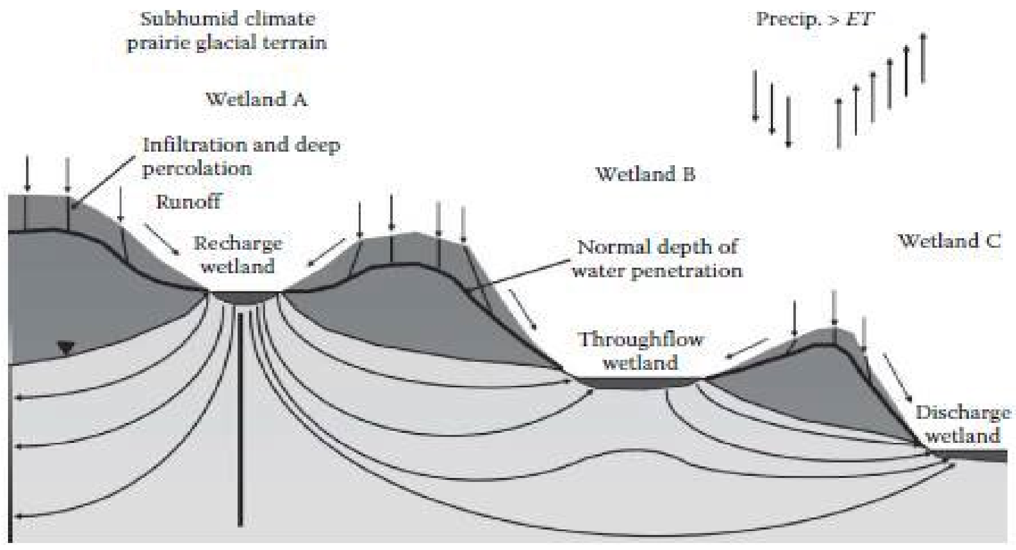
PPR650.1909 Drainage Equations for Lateral Effect Determination

B. Applicability to HGM Classes

(3) Drainage equations have been considered applicable only to MINERAL FLAT, ORGANIC FLAT, AND recharge DEPRESSIONAL wetland classes due to hydrodynamics being vertical (210-NEH-650.19). Surface storage is an applicable variable to vertical movement of water starting as rainfall, filling depressional storage created by surface micro-relief, then infiltrating, or running off along the surface to these wetland classes for ponding. Overland flow and snowmelt supply these wetland classes with relatively fresh water that recharges the groundwater with vertical movement of water leaching salts and leaving the upper layers considered non-saline. After this water infiltrates and "…flows in long groundwater flow paths, wetlands in the low areas receive substantial amounts of dissolved ions and other solutes from the higher areas" (Richardson, 2001); these wetlands receiving this high ionic strength water are considered discharge and flow through DEPRESSIONAL wetlands as well as SLOPE wetlands.

Figure PPR19-1: Landscape flow paths including depressional wetlands

Hydrology of Wetland and Related Soils 81



Subhumid climate prairie glacial terrain
Precip. > *ET*
Wetland A
Infiltration and deep percolation
Runoff
Recharge wetland
Wetland B
Normal depth of water penetration
Wetland C
Throughflow wetland
Discharge wetland

FIGURE 3.36
In subhumid landscapes, the groundwater divide is often in a depression. These landscapes often have flowthrough and discharge wetlands as well as recharge wetlands.

(4) SLOPE, discharge and flow through DEPRESSIONAL wetlands have strong effervescence and identify as alkaline or saline due to accumulation of ions and solutes from groundwater inflows. This hydrology provides unique soil taxonomy and habitat for plants, organisms, and animals that rely on alkalinity and salinity. SLOPE, discharge and flow through DEPRESSIONAL wetlands also vary in size and hydroperiod from recharge DEPRESSIONAL wetlands, which is typically larger in size with later and longer periods of saturation. To maintain these unique functions, drainage equation variables specifically related to SLOPE, discharge and flow through DEPRESSIONAL wetlands were adjusted.

(5) Drainage phenomenon still applicable to SLOPE, discharge and flow through DEPRESSIONAL wetlands using van Schilfgaarde equation include an initial saturated soil profile and water table fall applicable to soil conductivity and drain depth and size. Furthermore, the impermeable layer must be lower than the typical subsurface drain depth of 2-5 feet, in order not to intercept major flow path along an aquitard. In cases where hydraulic conductivity in the drain vicinity, considered depth of 2-5 feet, is one-tenth or less of the hydraulic conductivity above the drain a site-specific analysis is required to evaluate major diversion along an aquitard. For instances where hydraulic conductivity below the drain is greater than one-tenth of the hydraulic conductivity above the drain, drainage equations including van Schilfgaarde are applicable.

(6) The Corps of Engineers Wetlands Delineation Manual (USACE, 1987) defines ponded conditions where water stands and saturated conditions as voids filled with water to the soil surface. The manual describes wetlands with ground-water seepage as seldom, if ever, flooded, but soils saturated for a sufficient portion of the growing season. Adjusted drainable porosity is due to water "stored on the surface in depressions, so movement overland toward the drains may be restricted" (USDA, 1980). Since SLOPE, discharge and flow through DEPRESSIONAL wetlands are result of groundwater seepage, there is rarely ponded conditions resulting in water stored on the surface. In the same reference where surface storage value was adopted for MINERAL FLAT, ORGANIC FLAT, AND recharge DEPRESSIONAL wetland classes, surface storage was described as "Surface detention will temporarily sustain runoff when rainfall has ended" (Gilley, 2018). Both references highlight surface storage resulting from precipitation and overland runoff resulting in ponded conditions.

(7) Pali (2013) found the van Schilfgaarde equation to be applicable for saline and waterlogged soils, which describes discharge and flow-through wetlands. The Pali case study did not include surface storage adjustment, nor did the original van Schilfgaarde equation. Numerous hydrogeomorphic assessment approaches distinguish ~500 feet as ideal buffer zone for maintaining wetland functions. The USDA-NRCS functional assessment for Lake Dakota Sand Plains (USDA, 1999) "Hydrology Alterations" variable is optimized at a distance greater than 500 feet. Similarly, two USACE hydrogeomorphic guidebooks (USACE, 2011 & USACE, 2013) for applicable wetland classes also include an ideal buffer distance of 150 meters (492 feet). Therefore, the maximum lateral effect distant was capped at 500 feet in the PPR database.

(8) SLOPE, discharge and flow through DEPRESSIONAL wetlands endosaturation from groundwater and not from overland runoff. Since discharge and flow through DEPRESSIONAL wetlands predominant hydrology is from groundwater; water ponded or trapped on the surface within the wetland and surrounding the wetland by soil roughness seldomly occurs. Therefore, surface storage is not applicable and not recommended to be used in applying drainage equations, including van Schilfgaarde, to discharge and flow through DEPRESSIONAL wetlands. Application of drainage equations, including van Schilfgaarde without surface storage, was found to have appropriate setbacks for discharge and flow through DEPRESSIONAL wetlands analyzed with over twenty-five site specific analysis the NRCS has completed in the past. This association analysis found the most ideal correlation for discharge and flow through DEPRESSIONAL wetlands was with the original van Schilfgaarde equation, which was not modified with surface storage parameter.

(9) RIVERINE and LACUSTRINE FRINGE wetland classes require site specific analysis using appropriate field data as described in 210-NEH-650.19.

(10) Lateral effect distances were calculated with a Python script developed from ND-Drain computer program, as described in 210-NEH-650.19. Rosetta model version 1.3 is used for processing soils data. Supplemental information is provided for processing of organic soils data, and adoption of drainage equation and required parameters.

H. Processing of Organic Soils Data

(1) Organic soils are common in the PPR and are often found in hydric settings and are either coincident or proximal to wetlands. These organic soils are also often located on adjacent agricultural lands subject to agricultural drainage practices and thus it is important that lateral effect distances for these soils be accurately determined. However, the Pedo-Transfer Functions (PTF) used to derive the hydraulic parameters necessary for analytically producing lateral effect distances for mineral soils are not capable of predicting these properties for soils containing significant amounts of organic matter (Schapp et al., 2001). This is in part due to soil organic matter generally not having a granular textural composition like sand or silt, but rather have in lieu textural compositions that are fibrous to mucky. Thus, the PTFs that are calibrated to mineral, that is granular materials, are not calibrated to, or accurate for soils comprised of materials having fibrous or mucky in lieu textures. Furthermore, due to regional and legacy inconsistencies in the analyses and methods used to populate NASIS data for organic soils within the PPR, data contained in the database for these soils are continually in the process of being updated. However, at the time of this writing there are still some areas that contain anomalies and thus using these data from the database outright will produce inconsistencies in lateral effect distances. For these reasons, an alternate approach for determining the values of hydraulic properties and bulk densities for organic soils within the PPR was developed.

(2) This method is primarily based upon the representative in lieu texture of the organic material. The in lieu texture of an organic soil represents the decomposition and weathering of the at once plant material of which they were derived. Three in lieu texture classifications frequently used in NASIS to characterize soil horizons designated as organic are used: Peat, Mucky Peat, and Muck. These in lieu textures represent organic matter on a scale of presumed decomposition from Peat, slightly decomposed; to Mucky Peat, moderately decomposed; to Muck, highly decomposed. It is well documented that hydraulic properties and bulk densities of organic soils follow predictive trends as decomposition ranges from slight to high through these classifications. Soils containing more peat, or fibric, material are more porous, permeable, and less dense than soils containing more highly decomposed mucky, or sapric materials (Verry et al., 2011). Thus, it is often reasonable to assign properties to organic soils based on their level of decomposition (Verry et al., 2011; Soil Survey Staff, 2021; S.P.P. Grover, J.A. Baldock, 2013).

Figure PPR19-2: Primary organic textures and associated properties and descriptive information from NASIS (Soil Survey Staff, 2021).

Organic Texture	Decomposition	Common NASIS Texture Label	Common NASIS Horizon Label	Horizon Description
Peat	Slight	PEAT	Oi	Slightly decomposed organic material. The fiber content of these materials is 40 percent or more (by volume) after rubbing.
Mucky Peat	Intermediate	MPT	Oe	Organic material of intermediate decomposition. The fiber content of these materials is 17 to 40 percent (by volume) after rubbing.
Muck	Heavy	MUCK	Oa	Highly decomposed organic matter. Highly decomposed organic materials, which have a fiber content of less than 17 percent (by volume) after rubbing.

(3) To determine appropriate values of the hydraulic parameters required to calculate lateral effect distances in the PPR for these organic soils (saturated hydraulic conductivity, saturated water content, residual water content, and bulk densities), analysis and statistics of these

parameters as they are contained within the NASIS were performed on all organic horizons throughout the PPR (Soil Survey Staff, 2021). . The analysis and subsequent determination were done by querying these data for all organic horizons contained within soil components occupying greater than 10% of their respective map units within the states of IA, MN, ND, and SD (section PPR650.1916B). Note: minor components occupying less than 10% of the map unit area are not included in the determination of lateral effect distance for that map unit.

(4) Next an analysis of these results was performed to assess the distribution of horizon and in lieu textural classifications across the region. The analysis consisted of determining unique in lieu texture values, their frequency of occurrence throughout the PPR and among all organic soils (figure PPR19-2). This resulted in 16 unique in lieu texture identifiers among all the data, many of which were associated with relatively few horizons throughout the region. These 16 texture identifiers were then assigned into one of the three previously mentioned in lieu textures: peat, mucky-peat, muck; based on simple majority of the horizon name field. In lieu textures having most horizon names designated as slightly degraded organic material, or Oi, were assigned to peat. In lieu textures having most of the horizon names being designated as moderately degraded were assigned to mucky peat. Lastly, in lieu textures having most horizon names indicating heavily decomposed material, Oa, were assigned muck (figure PPR19-3).

Figure PPR19-3: Texture classes found in organic soils, the states they are contained within, horizon names, the frequency and percentage of all organic horizons, the associated primary texture class, and whether a texture was used to calculate mean property values of soil properties.

NASIS Texture	States w/ Texture	Horizon Names	Majority Horizon Name	States Containing Horizons	Number of Horizons	Percent of All Organic Horizons	Reclassified Texture	Used for Mean
CE	MN	Oa2	Oa	ALL	1	0.0%	MUCK	N
FLV-	MN	Oe	Oe	ALL	26	0.6%	MPT	N
HB-M	IA	Oa3	Oa	ALL	5	0.1%	MUCK	N
HPM	MN, SD	Oa	Oa	ALL	9	0.2%	MUCK	N
MARL	MN	Oma	Oa	ALL	1	0.0%	MUCK	N
MK-P	MN	Oe*1,2,3	Oe	MAJORITY	8	0.2%	MPT	N
MK-S	MN	Oa2	Oa	ALL	1	0.0%	MUCK	N
MPM	MN, ND, SD	O, Oi, Oe	Oe	MAJORITY	123	2.9%	MPT	**Y**
MPT	ALL	Oe*1,2,3	Oe	MAJORITY	1173	27.2%	MPT	**Y**
MUCK	ALL	Oe, Oa*1,2,3	Oa	MAJORITY	2237	51.9%	MUCK	**Y**
PEAT	MN, ND	Oi, Oe	Oi	MAJORITY	272	6.3%	PEAT	**Y**
S	MN	Oe, A	Oe	ALL	4	0.1%	MPT	N
SP	MN	Oa2	Oa	ALL	2	0.0%	MUCK	N
SPM	ND, SD	Oi, Oe, Oa	Oi	MAJORITY	440	10.2%	PEAT	**Y**
SR-	MN	Oa, C, Cg	Oa	MAJORITY	2	0.0%	MUCK	N
STX-	MN	Oe	Oe	ALL	7	0.2%	MPT	N

(5) The next step consisted of determining values of each hydraulic parameter to assign to each designated in lieu texture class. To accomplish this the mean values from a subset of horizons consisting only of those with in lieu texture classifications representing more than 1% of the

organic soils were used (figure PPR19-3). The mean values for each in lieu texture class and soil property combination are given in figure PPR19-4.

Figure PPR19-4: Average hydraulic soil parameters and bulk density for each primary organic textural class.

Texture Class	Saturated Hydraulic Conductivity	Bulk Density	Saturated Water Content	Residual Water Content
	(um/s)	(g/cm3)	(-)	(-)
MUCK	24.9	0.26	0.64	0.17
MPT	92.8	0.16	0.65	0.14
PEAT	295.5	0.11	0.67	0.09

(6) Lastly, the calculation of lateral effect distances requires the input of a value for drainable porosity. Values of this quantity are not stored in NASIS and are typically calculated using a soil water retention relationship. The van Genuchten (VG) relationship is commonly employed to calculate drainable porosity (van Genuchten, 1980). However, the Maualem-van Genuchten (VGM) model is better suited for high organic matter soils and requires derivation of two parameters, alpha and n, for this calculation (Liu and Lennertz, 2019). For mineral soils the values of these parameters are generated while running the texture based PTFs. Likewise, a similar approach may be used to derive the values of these two parameters for organic soils. Liu and Lennartz (2019) published PTFs for organic soils that require the primary input of bulk density and in the case of the alpha parameter mean horizon depth (equations 2 3). Drainable porosity can then be calculated for organic soils using equation 1 and the derived VGM parameters in conjunction with the previously mentioned mean class properties of saturated and residual water contents.

Equation 1: van Genuchten equation for soil water retention relationship (van Genuchten, 1980).

$$\theta(\psi) = \theta_r + \frac{\theta_s - \theta_r}{[1 + (\alpha|\psi|)^n]^{(1-\frac{1}{n})}}$$

Where:

$\theta(\psi)$: drainable porosity at the drawdown head (-)

θ_r: Residual water content (-)

θ_s: Saturated water content (-)

α: VG alpha parameter (1/cm)

ψ: drawdown head (cm)

n: VG n parameter (-)

Equation 2: Calculation of VGM alpha parameter as proposed by Liu and Lennartz, 2018

$$\alpha = 0.326 - 9.315\rho_b + 10.42{\rho_b}^2 - 0.014\bar{d}$$

Where:

α: VG alpha parameter (1/cm)

ρ_b: bulk density (g/cm^3)

$\bar{d}$: mean horizon depth (cm)

Equation 3: Calculation of VGM n parameter as proposed by Liu and Lennartz, 2018

$$n = 0.153 - 0.422\rho_b + 0.45{\rho_b}^2$$

Where:

n: VG n parameter (-)

ρ_b: bulk density (g/cm^3)

(7) In summary, when calculating a lateral effect distance for a soil component having one or more organic horizons the texture associated with each horizon was mapped to one of the primary classes as shown in Figure PPR19-3. Then hydraulic properties and bulk densities were assigned based on the primary texture class values in figure PPR19-4. The bulk density and mean horizon depth are then used to calculate the VGM parameters (equations 2, 3) and subsequently the drainable porosity (equation 1). After this, all the necessary inputs are available to calculate the lateral effect distance in the same manner as mineral soils.

I. Drainage Equation and Parameters.

(1) van Schilfgaarde equation (VS) was used for computation of lateral effects in the PPR. The primary reason for choosing this equation is that it is nonsteady and due to assumptions based on more sporadic than constant rainfall, which is relevant to PPR. Critical parameter assumptions are listed in bullets below.

(i) Surface Storage: As discussed previously in this document, surface storage for discharge and flow through DEPRESSIONAL wetlands is not applicable and therefore 0.0 inches. Surface storage experiments found average value in agricultural settings to be 0.12 inches (Gilley, 2018); which is applied to MINERAL FLAT, ORGANIC FLAT, AND recharge DEPRESSIONAL wetlands.

(ii) Drawdown Depth: The depth of drawdown from normal conditions is 12 inches, based on wetland hydrology indicators in USACE Regional Supplement (USACE, 2010).

(iii) Drawdown Time: The time of drawdown from normal conditions is 14 days, based on wetland hydrology indicators in USACE Regional Supplement (USACE, 2010).

(iv) Depth to Impermeable Layer: The depth to impermeable layer is 10 feet below the soil surface, based on typical conditions in PPR like national guidance that states, "The impermeable layer is generally assumed to occur at 10 feet below the soil surface" (210-NEH-650.19). This assumption is considered valid due to lack of roots and earthworm channels at this depth, as well as weight of soil compressing lower layers to reduce permeability. Fifty-five field and lab samples completed in PPR glacial till sites validated the 10-foot depth selection, as there is a slope change and order of magnitude drop in hydraulic conductivity at depths between 9 and 14 feet (van der Kamp, 2009).

(v) Hydraulic Conductivity below 2 meters: NASIS soil data exists between the ground surface and 2 meters (6.5 feet) of depth. Since the impermeable layer is 10 feet and calculations rely on the whole depth, hydraulic conductivity needs to be estimated. Soil conditions in the deepest NASIS layer, typically 1.5-2.0 feet, showed very consistent values through PPR soils. Therefore, the deepest soil horizon hydraulic conductivity was extrapolated, with no modification, to depth of 10 feet.

(vi) Drain Sizes: Typical drain size diameters range from 3-inches to 12-inches. Open ditches are also common near wetlands. Sensitivity to drain diameters is small, which means large increase in diameter has small increase in lateral effect distance. Drain tile manufacturers have varying available sizes. Therefore, the common agricultural drain sizes of 3, 4, 5, 6, 8, 10, 12-inch, and open ditch are available to choose from for evaluation. If the proposed drain diameter doesn't match one of the depths included in the database, users are advised to choose the next larger size.

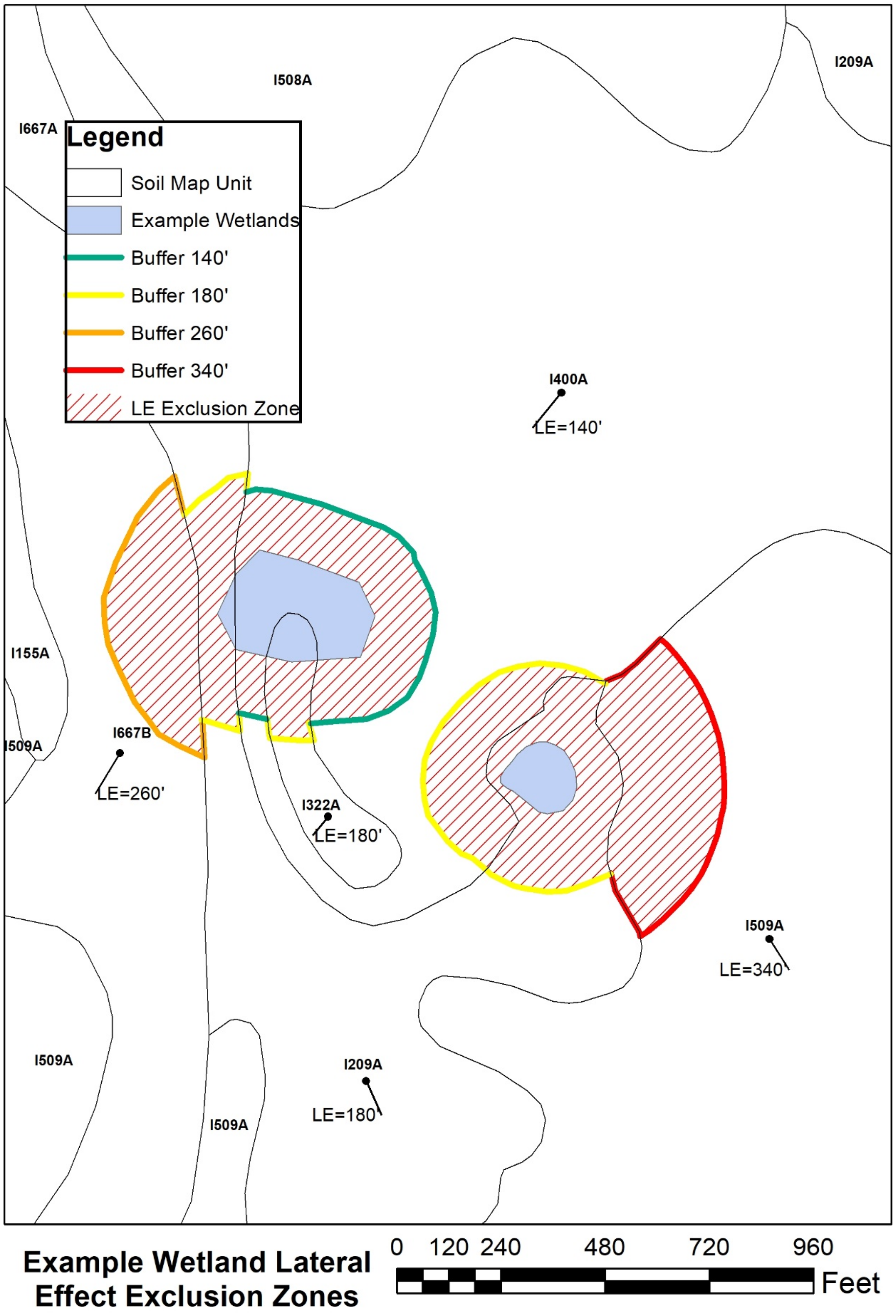
(vii) Drain Depths: Precise planned drain depths can be used directly. If a range of depths are planned for an area, users of the database are advised to enter the deepest depth in the range. The available drain depths to choose from are even foot increments including 2, 3, 4, 5, 6, 7, and 8 feet deep.

(viii) Components Composition in Soil Map Units: Each soil map unit includes multiple components, which vary in percent composition soil properties and qualities, and position on the landforms in the landscape. The soil map unit can be identified for the entire PPR using Web Soil Survey at https://websoilsurvey.nrcs.usda.gov/ (210-NEH-650.19). Components with less than 10 percent composition are not considered in LE calculations. All soil components equal or greater than 10% composition are considered. The highest value of these components is the calculated LE for the soil map unit.

(ix) Soil Map Unit Choice Considering multiple options: Calculate the lateral effect value for the wetland soil map unit, the soil map unit where the tile will be placed, and the soil map unit of the soil between the tile and the wetland boundary (if applicable). Apply the highest lateral effect value calculated and draw drain tile exclusion zone following buffered distance and soil map unit boundaries. If drain tile is only on one side of wetland, the opposite side does not need to be calculated or considered. Two example wetland lateral effect exclusion zones are shown in figure 3, for respective side a drain tile is planned. The stand-alone distance can be used, or spatial drawing with exclusion zone if producer has that capability. West wetland has three distinct lateral effect distances, i.e., 140, 180, and 260 feet. East wetland has two distinct lateral effect distances, i.e., 180 and 340 feet. Sharp transitions can occur at soil map unit lines that replicate "sawtooths", which include area outside lateral effect exclusion zone. Installing drains in these sawtooths can slightly short circuit the longer lateral effect. Therefore, caution should be exercised installing drainage in these sawtooths.

(2) For more complex sites, with more soil map units within wetland extents or within tile extents, request assistance from NRCS wetland compliance or engineering staff.

Figure PPR19-5: Example Wetland Lateral Effect Exclusion Zones



(PPR210-650-H, Mar 2022)

PPR650.1914 References

Z. Brinson, M.M. 1993. A hydrogeomorphic classification for wetlands. Technical Report WRP–DE–4. U.S. Army Corps of Engineers Waterways Experiment Station, Vicksburg, MS.

AA. Gilley, J.E., 2018. Surface Detention on Cropland, Rangeland, and Conservation Reserve Program Areas. Biological Systems Engineering: Papers and Publications. 562. http://digitalcommons.unl.edu/biosysengfacpub/562.

BB. Grover, S.P.P., Baldock, J.A., 2013. The link between peat hydrology and decomposition: Beyond von Post, Journal of Hydrology, Volume 479, 2013, Pages 130-138, ISSN 0022-1694,https://doi.org/10.1016/j.jhydrol.2012.11.049.

CC. Liu H. and Lennartz B., 2019. Hydraulic properties of peat soils along a bulk density gradient – A meta study. Hydrologic Process. 2019; 33:101-144. https://doi.org/10.1002/hyp.13314

DD. Schaap, M.G., Leij, F.J., van Genuchten, M.T., 2001. rosetta: a computer program for estimating soil hydraulic parameters with hierarchical pedotransfer functions, Journal of Hydrology, Volume 251, Issues 3–4, 2001, Pages 163-176, ISSN 0022-1694, https://doi.org/10.1016/S0022-1694(01)00466-8.

EE. Pali, A.K., 2013. Evaluation of Non Steady Subsurface Drainage Equations for Heterogeneous Saline Soils: A Case Study, IOSR-JAVS e-ISSN: 2319-2380, p-ISSN: 2319-2372. Volume 6, Issue 5 (Nov. - Dec. 2013), PP 45-52.

FF. Richardson, J.L. and Vepraskas, M.J. 2001. Wetland Soils, Genesis, Hydrology, Landscapes, and Classification.

GG. Smith, R.D., A. Ammann, C. Bartoldus, and M.M. Brinson. 1995. An approach for assessing wetland functions using hydrogeomorphic classification, reference wetlands, and functional indices. U.S. Army Engineer Waterways Experiment Station Technical Report WRP-DE-9, Vicksburg, MS.

HH. United States Army Corps of Engineers, 1987. Corps of Engineers Wetlands Delineation Manual, Technical Report Y-87-1, U.S. Army Engineer Waterways Experiment Station, Vicksburg, MS.

II. United States Army Corps of Engineers, 2010. Regional Supplement to the Corps of Engineers Wetland Delineation Manual: Great Plains Region (Version 2.0), U.S. Army Engineer Waterways Experiment Station, Vicksburg, MS.

JJ. United States Army Corps of Engineers, 2011. Regional Guidebook for Applying the Hydrogeomorphic Approach to Assessing the Functions of Headwater Slope Wetlands on the South Carolina Coastal Plain, ERDC/EL TR-11-11, September 2011.

KK. United States Army Corps of Engineers, 2013. Regional Guidebook for Applying the Hydrogeomorphic Approach to Assessing the Functions of Flat and Seasonally Inundated Depression Wetlands on the Highland Rim, ERDC/EL TR-13-12, June 2013.

LL. United States Department of Agriculture, Natural Resources Conservation Service. 1980. DRAINMOD Reference Report, Chapter 2.

MM. United States Department of Agriculture, Natural Resources Conservation Service. 1999. Interim Functional Assessment Model for Lake Dakota Sand Plains, Version 2.2, Prairie Pothole Wetland Team, Jamestown, ND, February 28, 1999.

NN. United States Department of Agriculture, Natural Resources Conservation Service. 2008. Hydrogeomorphic wetland classification system: An overview and modification to better meet the needs of the Natural Resources Conservation Service. Technical Note No. 190–8–76.

OO. U.S. Department of Agriculture, Natural Resources Conservation Service., 2012. 77 FR 12234: Changes In Hydric Soils Database Selection Criteria. pp. 12,234–12,235.

PP. United States Department of Agriculture, Natural Resources Conservation Service., 2014. Keys To Soil Taxonomy, 12th Edition. USDA–NRCS, Washington, DC.

QQ. United States Department of Agriculture, Natural Resources Conservation Service. 2018. Field Indicators of Hydric Soils in the United States, Version 8.2. L.M. Vasilas, G.W. Hurt, and J.F. Berkowitz (eds.). USDA, NRCS in cooperation with the National Technical Committee for Hydric Soils.

RR. United States Department of Agriculture, Natural Resources Conservation Service., 2021. Hydrology Tools For Wetland Identification And Analysis, Title 210, National Engineering Handbook, Part 650, Chapter 19, February 2021.

SS. United States Department of Agriculture, Natural Resources Conservation Service, 2021. National Soil Survey Handbook, Title 430, Amendment 39, May 2021.

TT. United States Department of Agriculture, Natural Resources Conservation Service., 2021. Web Soil Survey. Available online at the following link: https://websoilsurvey.sc.egov.usda.gov/. Accessed [3/4/2021].

UU. Van der Kamp, G., Hayashi, M., 2009. Groundwater-Wetland Ecosystem Interaction In The Semiarid Glaciated Plains Of North America, Hydrogeology Journal 17, 203-214.

VV. van Genuchten, M. Th., 1980. A Closed-Form Equation For Predicting The Hydraulic Conductivity Of Unsaturated Soils. Soil Sci. Soc. Am. J. 44:892-898.

WW. Vepraskas, M.J. and Craft, C.B., 2016. Wetland Soils, Genesis, Hydrology, Landscapes, and Classification Second Edition.

XX. Verry, Elon S.; Boelter, Don H.; Paivanen, J.; Nichols, Dale S.; Malterer, T.; Gafni, A., 2011. Physical Properties Of Organic Soils. In: Kolka, Randall K.; Sebestyen, Stephen D.; Verry, Elon S.; Brooks, Kenneth N., eds. Peatland Biogeochemistry And Watershed Hydrology At The Marcell Experimental Forest. Boca Raton, FL: CRC Press: 135-176.

PPR650.1916 Exhibits

A. HGM Wetland Class Scripts used in NASIS

HGM Wetland Class - Prairie Pothole Region

```
PARAMETER variable1 CHARACTER PROMPT "Area Symbol (e.g. ND001 OR ND*)".
BASE TABLE component.
EXEC SQL
SELECT areasymbol, liid, nationalmusym, musym, muiid, mustatus, lmapunitiid mukey, dmuiid,
    coiid, compname, localphase, comppct_r, hydricrating
FROM REAL area, REAL legend, lmapunit, REAL mapunit, correlation, REAL datamapunit,
    component
WHERE
areasymbol IMATCHES variable1 AND
JOIN area TO legend AND
JOIN legend TO lmapunit AND
JOIN lmapunit TO mapunit AND
JOIN mapunit TO correlation AND
JOIN correlation TO datamapunit AND
JOIN datamapunit TO component AND

#Option to remove the "PARAMETER" for single execution of this report on the 4 target PPR
    states.
#   (area.areasymbol IMATCHES "ND*" OR
#   area.areasymbol IMATCHES "SD*" OR
#   area.areasymbol IMATCHES "MN*" OR
#   area.areasymbol IMATCHES "IA*") AND

    legend.legendsuituse IN ("current wherever mapped") AND
    correlation.repdmu IN ("yes") AND
    lmapunit.mustatus IN ("correlated") AND
    component.hydricrating IN ("yes");
    SORT BY liid, musym SYM, comppct_r DESC, compname LEX.

EXEC SQL
SELECT lsgd.geomfeatid, lsft.geomftname, lsgf.geomfnamep
FROM component, OUTER (cogeomordesc lsgd), REAL geomorfeattype lsft, REAL geomorfeat
    lsgf
WHERE
JOIN component TO lsgd AND
JOIN lsgd TO lsgf AND
JOIN lsft TO lsgf AND
    lsft.geomftname IN ("Landscape") AND
    lsgd.rvindicator IN ("yes");
SORT BY lsgd.geomfeatid DESC
AGGREGATE COLUMN lsft.geomftname LAST, lsgf.geomfnamep LAST.

EXEC SQL
SELECT lfgd.geomfeatid, lfft.geomftname, lfgf.geomfnamep, shapeacross, shapedown
```

```
FROM component, OUTER (cogeomordesc lfgd), REAL geomorfeattype lfft, REAL geomorfeat
     lfgf, cosurfmorphss
WHERE
JOIN component TO lfgd AND
JOIN lfgd TO lfgf AND
JOIN lfft TO lfgf AND
JOIN lfgd TO cosurfmorphss AND
     lfft.geomftname IN ("Landform") AND
     lfgd.rvindicator IN ("yes");
SORT BY lfgd.geomfeatid DESC
AGGREGATE COLUMN lfft.geomftname LAST, lfgf.geomfnamep LAST.

DERIVE sodium1          FROM low USING "MLRA10_StPaul" : "SAR MAX IN
     DEPTH 0-50 CM BELOW DUFF, ABOVE RESTRICTION (ND)".
DERIVE gyp              FROM low USING "MLRA10_StPaul" : "GYP MAX IN
     DEPTH 0-50 CM BELOW DUFF, ABOVE RESTRICTION (ND)".
DERIVE charge           FROM low USING "MLRA10_StPaul" : "EC MAX IN DEPTH
     0-50 CM BELOW DUFF, ABOVE RESTRICTION (ND)".
DERIVE lime       FROM low USING "MLRA10_StPaul" : "CaCO3 MAX IN DEPTH 0-
     50 CM BELOW DUFF, ABOVE RESTRICT (ND)".
DERIVE flood1     FROM rv USING "MLRA10_StPaul" : "FLOODING FREQUENCY
     MAX GROWING SEASON (ND)".
DERIVE histic1          FROM rv USING "MLRA10_StPaul" : "HISTIC EPIPEDON
     THICKNESS (ND)".

DEFINE cname      ISNULL (localphase) THEN compname ELSE cname||", "||localphase.
ASSIGN hydricrating CODENAME(hydricrating).

#Sodium.
DEFINE sodium                 IF (sodium1 >= 5) THEN "sodic" ELSE "not sodic".

#Histic epipedon.
DEFINE histic           IF (histic1 >= 20) THEN "organic" ELSE "mineral".

#Carbonates, gypsum, and other soluable salts.
DEFINE salt                   IF (((lime >= 1) OR (gyp >= 1) OR (charge >= 4.001))
                              OR (localphase IMATCHES "*saline*") OR (compname
     MATCHES "Salt flats")) THEN "present" ELSE "absent".

#Floods during growing season and on flooded landform.
DEFINE flood            IF (((flood1 MATCHES "very frequent") OR (flood1
     MATCHES "frequent") OR (flood1 MATCHES "occasional") OR (flood1 MATCHES
     "rare")) AND
                              ((lfgf.geomfnamep MATCHES "flood plains") OR
     (lfgf.geomfnamep MATCHES "flood-plain steps") OR (lfgf.geomfnamep MATCHES
     "oxbows") OR
                              (lfgf.geomfnamep MATCHES "drainageways") OR
     (lfgf.geomfnamep MATCHES "channels"))) THEN "yes" ELSE "no".

#Define fen HGM wetland class.
```

```
DEFINE wetland_class  IF ((sodium MATCHES "not sodic") AND (histic MATCHES "organic")
     AND (flood MATCHES "no") AND (lfgf.geomfnamep MATCHES "fens")) THEN "fen"
     ELSE "".

#Define organic flat HGM wetland class.
ASSIGN wetland_class  IF (((wetland_class == "") AND (wetland_class != "fen")) AND (sodium
     MATCHES "not sodic") AND (histic MATCHES "organic") AND (flood MATCHES "no"))
                                             THEN "organic flat" ELSE wetland_class.

#Define riverine HGM wetland class.
ASSIGN wetland_class  IF ((((wetland_class == "") AND (wetland_class != "fen") AND
     (wetland_class != "organic_flat")) AND
                                             (sodium MATCHES "not sodic") AND (histic
     MATCHES "mineral") AND (flood MATCHES "yes")) AND
                                             ((lfgf.geomfnamep MATCHES "flood plains") OR
                                             (lfgf.geomfnamep MATCHES "flood-plain steps") OR
                                             (lfgf.geomfnamep MATCHES "oxbows") OR
                                             (lfgf.geomfnamep MATCHES "abandoned channels")
     OR
                                             (lfgf.geomfnamep MATCHES "drainageways") OR
                                             (lfgf.geomfnamep MATCHES "channels")))
                                             THEN "riverine" ELSE wetland_class.


#Define mineral flat HGM wetland class.
ASSIGN wetland_class  IF ((((wetland_class == "") AND (wetland_class != "fen") AND
     (wetland_class != "organic_flat") AND (wetland_class != "riverine")) AND
                                             (sodium MATCHES "not sodic") AND (histic
     MATCHES "mineral") AND (salt MATCHES "absent") AND (flood MATCHES "no"))
     AND
                                             (lfgf.geomfnamep MATCHES "flats"))
#                                            (lfgf.geomfnamep MATCHES "lake plains") OR
#                                            (lfgf.geomfnamep MATCHES "deltas") OR
#                                            (lfgf.geomfnamep MATCHES "till-floored lake plains")
     OR
#                                            (lfgf.geomfnamep MATCHES "collapsed ice-walled
     lakebeds")))
                                             THEN "mineral flat" ELSE wetland_class.

#Define recharge HGM wetland class.
ASSIGN wetland_class  IF ((((wetland_class == "") AND (wetland_class != "fen") AND
     (wetland_class != "organic_flat") AND (wetland_class != "riverine") AND
                                             (wetland_class != "mineral_flat")) AND
                                             (sodium MATCHES "not sodic") AND (histic
     MATCHES "mineral") AND (salt MATCHES "absent") AND (flood MATCHES "no"))
     AND
                                             ((lfgf.geomfnamep MATCHES "depressions") OR
#                                            (lfgf.geomfnamep MATCHES "stream terraces") OR
                                             (lfgf.geomfnamep MATCHES "sand sheets") OR
#                                            (lfgf.geomfnamep MATCHES "moraines") OR
                                             (lfgf.geomfnamep MATCHES "potholes") OR
```

```
                                            (lfgf.geomfnamep MATCHES "playas") OR
                                            (lfgf.geomfnamep MATCHES "drainageways")))
                                            THEN "recharge" ELSE wetland_class.

#Define discharge HGM wetland class.
ASSIGN wetland_class IF (((((wetland_class == "") AND (wetland_class != "fen") AND
    (wetland_class != "organic_flat") AND (wetland_class != "riverine") AND
                                            (wetland_class != "mineral_flat") AND (wetland_class
    != "recharge")) AND
                                            (sodium MATCHES "not sodic") AND (histic
    MATCHES "mineral") AND (salt MATCHES "present") AND (flood MATCHES "no"))
    AND
                                            ((lfgf.geomfnamep MATCHES "depressions") OR
                                            (lfgf.geomfnamep MATCHES "rims") OR
#                                           (lfgf.geomfnamep MATCHES "terraces") OR
                                            (lfgf.geomfnamep MATCHES "playas") OR
                                            (lfgf.geomfnamep MATCHES "potholes") OR
                                            (lfgf.geomfnamep MATCHES "marshes") OR
#                                           (lfgf.geomfnamep MATCHES "deltas") OR
#                                           (lfgf.geomfnamep MATCHES "delta plains") OR
#                                           (lfgf.geomfnamep MATCHES "outwash plains") OR
#                                           (lfgf.geomfnamep MATCHES "lakeshores") OR
#                                           (lfgf.geomfnamep MATCHES "lake plains") OR
#                                           (lfgf.geomfnamep MATCHES "till-floored lake plains")
    OR
#                                           (lfgf.geomfnamep MATCHES "collapsed ice-walled
    lakebeds") OR
#                                           (lfgf.geomfnamep MATCHES "beaches") OR
                                            (lfgf.geomfnamep MATCHES "drainageways") OR
                                            (lfgf.geomfnamep MATCHES "flats"))) OR (compname
    MATCHES "Salt flats"))
                                            THEN "discharge" ELSE wetland_class.

#Define slope HGM wetland class.
ASSIGN shapeacross          CODENAME(shapeacross).
ASSIGN shapedown            CODENAME(shapedown).
ASSIGN wetland_class IF ((((wetland_class == "") AND (wetland_class != "fen") AND
    (wetland_class != "organic_flat") AND (wetland_class != "riverine") AND
                                            (wetland_class != "mineral_flat") AND (wetland_class
    != "recharge") AND (wetland_class != "discharge")) AND
                                            (sodium MATCHES "not sodic") AND (histic
    MATCHES "mineral") AND (salt MATCHES "present") AND (flood MATCHES "no"))
    AND
                                            (((shapedown MATCHES "concave") AND
    (shapeacross MATCHES "concave") OR
                                            (shapedown MATCHES "concave") AND (shapeacross
    MATCHES "linear") OR
                                            (shapedown MATCHES "concave") AND (shapeacross
    MATCHES "convex") OR
                                            (shapedown MATCHES "convex") AND (shapeacross
    MATCHES "concave") OR
```

```
                                    (shapedown MATCHES "linear") AND (shapeacross
    MATCHES "concave")) OR
                                    (lfgf.geomfnamep MATCHES "seeps")))
                                    THEN "slope" ELSE wetland_class.

#Exclude sodium affected soils. Sodium alters the hydrology via dispersion and its affects on
    water infiltration require an on-site investigation.
ASSIGN wetland_class IF (((wetland_class == "") AND (wetland_class != "fen") AND
    (wetland_class != "organic_flat") AND (wetland_class != "riverine") AND
                                    (wetland_class != "mineral_flat") AND (wetland_class
    != "recharge") AND (wetland_class != "discharge") AND (wetland_class != "slope")) AND
                                    (sodium MATCHES "sodic")) THEN "sodic" ELSE
    wetland_class.

PAGE WIDTH UNLIMITED LENGTH UNLIMITED.

TEMPLATE alpha SEPARATOR "|"
    AT LEFT
            FIELD WIDTH UNLIMITED SEPARATOR "",
            FIELD WIDTH UNLIMITED,
            FIELD WIDTH UNLIMITED,
            FIELD WIDTH UNLIMITED,
            FIELD WIDTH UNLIMITED,
            FIELD WIDTH UNLIMITED,
            FIELD WIDTH UNLIMITED,
            FIELD WIDTH UNLIMITED,
            FIELD WIDTH UNLIMITED,
            FIELD WIDTH UNLIMITED,
            FIELD WIDTH UNLIMITED,
            FIELD WIDTH UNLIMITED,
            FIELD WIDTH UNLIMITED,
            FIELD WIDTH UNLIMITED,
            FIELD WIDTH UNLIMITED,
            FIELD WIDTH UNLIMITED,
            FIELD WIDTH UNLIMITED,
            FIELD WIDTH UNLIMITED.

SECTION A
    HEADING
    USING alpha
        "areasymbol",
        "nationalmusym",
        "musym",
        "compname",
        "localphase",
        "comppct",
        "hydric",
        "LSgeomorfeattype",
        "LSgeomfnamep",
        "LFgeomorfeattype",
        "LFgeomfnamep",
        "sodium",
        "histic",
        "salts",
        "floods?",
        "wetland_class",
        "coiid",
        "mukey".
END SECTION.

SECTION B
    DATA
    USING alpha
        areasymbol,
        nationalmusym,
        musym,
        compname,
        localphase,
        comppct_r,
        hydricrating,
        lsft.geomftname,
        lsgf.geomfnamep,
        lfft.geomftname,
        lfgf.geomfnamep,
        sodium,
        histic,
        salt,
        flood,
        wetland_class,
        coiid NO COMMA ,
        mukey NO COMMA .

END SECTION.
```

SAR MAX IN DEPTH 0-50 CM BELOW DUFF, ABOVE RESTRICTION (ND)

```
BASE TABLE component.

# Get maximum SAR in any horizon in 0-50 cm range below organic duff and above restriction.
EXEC SQL
SELECT hzdept_r, sar_l, sar_r, sar_h
FROM  component, chorizon
WHERE
JOIN component TO chorizon;
SORT BY hzdept_r
AGGREGATE COLUMN hzdept_r NONE, sar_l NONE, sar_r NONE, sar_h NONE.

#Determine the thickness of organic duff and depth to RESTRICTIVE LAYER.
DERIVE o_thickness  FROM rv USING "MLRA10_StPaul":"SURFACE ORGANIC DUFF
    HORIZONS THICKNESS (ND)".
DERIVE depth            FROM rv USING "NSSC Pangaea":"DEPTH TO FIRST
    RESTRICTIVE LAYER".

ASSIGN sar_l IF hzdept_r > o_thickness + 50 OR hzdept_r < o_thickness THEN NULL ELSE
    sar_l.
ASSIGN sar_h IF hzdept_r > o_thickness + 50 OR hzdept_r < o_thickness THEN NULL ELSE
    sar_h.
ASSIGN sar_r IF hzdept_r > o_thickness + 50 OR hzdept_r < o_thickness THEN NULL ELSE
    sar_r.

#Find minimum of restriction depth and 50cm
DEFINE min_depth      depth < 51 AND NOT ISNULL(depth)? depth : 51.
DEFINE in_range       ISNULL(hzdept_r)? hzdept_r : ((hzdept_r < min_depth)? 1 : 0).

#Find the sodium adsorption ratio values in the min_depth.
DEFINE low                    ARRAYMAX(LOOKUP(1, in_range, sar_l)).
DEFINE high           ARRAYMAX(LOOKUP(1, in_range, sar_h)).
DEFINE rv             ARRAYMAX(LOOKUP(1, in_range, sar_r)).
```

GYP MAX IN DEPTH 0-50 CM BELOW DUFF, ABOVE RESTRICTION (ND)

```
BASE TABLE component.

# Get maximum gypsum in any horizon in 0-50 cm range below organic duff and above
    restriction.
EXEC SQL
SELECT hzdept_r, gypsum_l, gypsum_r, gypsum_h
FROM  component, chorizon
WHERE
JOIN component TO chorizon;
SORT BY hzdept_r
AGGREGATE COLUMN hzdept_r NONE, gypsum_l NONE, gypsum_r NONE, gypsum_h
    NONE.

#Determine the thickness of organic duff and depth to RESTRICTIVE LAYER.
DERIVE o_thickness  FROM rv USING "MLRA10_StPaul":"SURFACE ORGANIC DUFF
    HORIZONS THICKNESS (ND)".
DERIVE depth            FROM rv USING "NSSC Pangaea":"DEPTH TO FIRST
    RESTRICTIVE LAYER".

ASSIGN gypsum_l IF hzdept_r > o_thickness + 50 OR hzdept_r < o_thickness THEN NULL
    ELSE gypsum_l.
ASSIGN gypsum_h IF hzdept_r > o_thickness + 50 OR hzdept_r < o_thickness THEN NULL
    ELSE gypsum_h.
ASSIGN gypsum_r IF hzdept_r > o_thickness + 50 OR hzdept_r < o_thickness THEN NULL
    ELSE gypsum_r.

#Find minimum of restriction depth and 50cm
DEFINE min_depth      depth < 51 AND NOT ISNULL(depth)? depth : 51.
DEFINE in_range       ISNULL(hzdept_r)? hzdept_r : ((hzdept_r < min_depth)? 1 : 0).

#Find the gypsum values in the min_depth.
DEFINE low                    ARRAYMAX(LOOKUP(1, in_range, gypsum_l)).
DEFINE high           ARRAYMAX(LOOKUP(1, in_range, gypsum_h)).
DEFINE rv             ARRAYMAX(LOOKUP(1, in_range, gypsum_r))
```

<u>EC MAX IN DEPTH 0-50 CM BELOW DUFF, ABOVE RESTRICTION (ND)</u>

```
BASE TABLE component.

# Get maximum EC in any horizon in 0-50 cm range below organic duff and above restriction.
EXEC SQL
SELECT hzdept_r, ec_l, ec_r, ec_h
FROM  component, chorizon
WHERE
JOIN component TO chorizon;
SORT BY hzdept_r
AGGREGATE COLUMN hzdept_r NONE, ec_l NONE, ec_r NONE, ec_h NONE.

#Determine the thickness of organic duff and depth to RESTRICTIVE LAYER.
DERIVE o_thickness  FROM rv USING "MLRA10_StPaul":"SURFACE ORGANIC DUFF
    HORIZONS THICKNESS (ND)".
DERIVE depth            FROM rv USING "NSSC Pangaea":"DEPTH TO FIRST
    RESTRICTIVE LAYER".

ASSIGN ec_l IF hzdept_r > o_thickness + 50 OR hzdept_r < o_thickness THEN NULL ELSE
    ec_l.
ASSIGN ec_h IF hzdept_r > o_thickness + 50 OR hzdept_r < o_thickness THEN NULL ELSE
    ec_h.
ASSIGN ec_r IF hzdept_r > o_thickness + 50 OR hzdept_r < o_thickness THEN NULL ELSE
    ec_r.

#Find minimum of restriction depth and 50cm
DEFINE min_depth      depth < 51 AND NOT ISNULL(depth)? depth : 51.
DEFINE in_range       ISNULL(hzdept_r)? hzdept_r : ((hzdept_r < min_depth)? 1 : 0).

#Find the electrical conductivity values in the min_depth.
DEFINE low                    ARRAYMAX(LOOKUP(1, in_range, ec_l)).
DEFINE high           ARRAYMAX(LOOKUP(1, in_range, ec_h)).
DEFINE rv             ARRAYMAX(LOOKUP(1, in_range, ec_r)).
```

CaCO3 MAX IN DEPTH 0-50 CM BELOW DUFF, ABOVE RESTRICT (ND)

```
BASE TABLE component.

# Get maximum CaCO3 in any horizon in 0-50 cm range below organic duff and above
    restriction.
EXEC SQL
SELECT hzdept_r, CaCO3_l, CaCO3_r, CaCO3_h
FROM  component, chorizon
WHERE
JOIN component TO chorizon;
SORT BY hzdept_r
AGGREGATE COLUMN hzdept_r NONE, CaCO3_l NONE, CaCO3_r NONE, CaCO3_h
    NONE.

#Determine the thickness of organic duff and depth to RESTRICTIVE LAYER.
DERIVE o_thickness  FROM rv USING "MLRA10_StPaul":"SURFACE ORGANIC DUFF
    HORIZONS THICKNESS (ND)".
DERIVE depth             FROM rv USING "NSSC Pangaea":"DEPTH TO FIRST
    RESTRICTIVE LAYER."

ASSIGN CaCO3_l IF hzdept_r > o_thickness + 50 OR hzdept_r < o_thickness THEN NULL
    ELSE CaCO3_l.
ASSIGN CaCO3_h IF hzdept_r > o_thickness + 50 OR hzdept_r < o_thickness THEN NULL
    ELSE CaCO3_h.
ASSIGN CaCO3_r IF hzdept_r > o_thickness + 50 OR hzdept_r < o_thickness THEN NULL
    ELSE CaCO3_r.

#Find minimum of restriction depth and 50cm
DEFINE min_depth      depth < 51 AND NOT ISNULL(depth)? depth : 51.
DEFINE in_range       ISNULL(hzdept_r)? hzdept_r : ((hzdept_r < min_depth)? 1 : 0).

#Find the calcium carbonate equivalent values in the min_depth.
DEFINE low                    ARRAYMAX(LOOKUP(1, in_range, CaCO3_l)).
DEFINE high           ARRAYMAX(LOOKUP(1, in_range, CaCO3_h)).
DEFINE rv             ARRAYMAX(LOOKUP(1, in_range, CaCO3_r)).
```

FLOODING FREQUENCY MAX GROWING SEASON (ND)

```
BASE TABLE component.

# Get the longest flooding frequency class.
EXEC SQL
SELECT flodfreqcl
FROM component, comonth
WHERE
JOIN component TO comonth AND
(flodfreqcl NOT IN("none")) AND
((taxtempregime IN ("mesic") AND
    comonth.month IN("mar", "apr", "may", "jun", "jul", "aug", "sep", "oct")) OR
(taxtempregime IN("frigid") AND
    comonth.month IN("apr", "may", "jun", "jul", "aug", "sep")));
AGGREGATE COLUMN flodfreqcl UNIQUE.

DEFINE flooding CODENAME(flodfreqcl).
DEFINE duration flooding MATCHES "very frequent" ? 1 : flooding MATCHES "frequent" ? .8
    : flooding MATCHES "occasional" ? .6 : flooding MATCHES "rare" ? .4 : flooding
    MATCHES "very rare" ? .2 : flooding MATCHES "none" ? .1 : 1/0.
DEFINE longest ARRAYMAX(duration).
DEFINE rv LOOKUP (longest, duration, flooding).
```

HISTIC EPIPEDON THICKNESS (ND)

```
BASE TABLE component.
EXEC SQL
SELECT hzdepb_l low, hzdepb_r rv, hzdepb_h high
FROM component
INNER JOIN chorizon ON chorizon.coiidref=component.coiid
INNER JOIN chtexturegrp ON chtexturegrp.chiidref=chorizon.chiid AND
    chtexturegrp.rvindicator = 1
INNER JOIN chtexture ON chtexture.chtgiidref=chtexturegrp.chtgiid
AND ((desgnmaster IN ("O", "O'", "O''")) OR (hzname IN ("O*"))) AND
((lieutex IN ("mpm")) OR
(lieutex IN ("mpt")) OR
(lieutex IN ("muck")) OR
(lieutex IN ("peat")) OR
(lieutex IN ("spm")) OR
(lieutex IN ("udom")) OR
(lieutex IN ("pdom")) OR
(lieutex IN ("hpm")));

AGGREGATE COLUMN low MAX, rv MAX, high MAX.

ASSIGN low IF ISNULL(low) THEN 0 ELSE low.
ASSIGN rv IF ISNULL(rv) THEN 0 ELSE rv.
ASSIGN high IF ISNULL(high) THEN 0 ELSE high.
```

B. NASIS Query used to Acquire Organic Soil Properties

Data acquired from: https://SDMDataAccess.sc.egov.usda.gov/Tabular/post.rest on 3/4/2021

```
SELECT legend.areasymbol, mapunit.mukey, mapunit.musym, component.cokey,
component.compname, component.comppct_r, chorizon.chkey, chorizon.hzname,
chtexturegrp.texture,
chorizon.hzdept_r, chorizon.hzdepb_r,
ISNULL(chorizon.ksat_r, -9.9) as 'ksat_r',
ISNULL(chorizon.om_r, -9.9) as 'om_r',
ISNULL(chorizon.dbthirdbar_r, -9.9) as 'dbthirdbar_r',
ISNULL(chorizon.wsatiated_r, -9.9)/100.0 as 'wsatiated_r',
ISNULL(chorizon.wthirdbar_r, -9.9)/100.0 as 'wthirdbar_r',
ISNULL(chorizon.wfifteenbar_r, -9.9)/100.0 as 'wfifteenbar_r'
FROM (legend INNER JOIN (mapunit
INNER JOIN component ON mapunit.mukey = component.mukey)
ON legend.lkey = mapunit.lkey)
INNER JOIN chorizon ON component.cokey = chorizon.cokey
INNER JOIN chtexturegrp ON chorizon.chkey = chtexturegrp.chkey
WHERE ((legend.areasymbol like 'IA%') OR (legend.areasymbol like 'MN%')
OR (legend.areasymbol like 'ND%') OR (legend.areasymbol like 'SD%')
AND (component.comppct_r>=10)
AND (chtexturegrp.rvindicator = 'Yes') AND (mapunit.musym <> 'W')
AND (chorizon.hzname LIKE 'Oi'))
ORDER BY mapunit.mukey, component.cokey, chorizon.hzdept_r
```

C. Lateral Effect Calculation Script

```
# -*- coding: utf-8 -*-
"""
Created on Tue Dec 15 12:48:34 2020

@author: Jason.Roth
@title: Environmental Engineer WQQT
@affiliation: USDA-NRCS WNTSC
@email:jason.roth@usda.gov



DESC: Script containing suite of functions and calls to produce
a lateral effect distance for the purposes of wetlands conservation

"""
import os
import requests as req
import numpy as np
import math
import sys



def make_file_hdr(names, units=0, delim=","):
    """
    reads in a tab delimited data file

    Parameters
    ----------
    names : list, list of strings of header names
    units : units, list of strings of corresponding units
    delim : str, delimeter character

    Returns
    -------
    h : str, containing file header.
    """

    h = str(names[0])
    names.pop(0)
    for n in names:
        h+="{0} {1}".format(delim, n)
    if len(units)>=len(names):
        h+="\n{0}".format(str(units[0]))
        units.pop(0)
        for u in units:
            h+="{0} {1}".format(delim, u)
    return h

def read_data_file(file_path, skp_lin=0):
    """
    reads in a tab delimited data file

    Parameters
    ----------
    file_path : str, path to data file
    skp_lin : int, number of lines to skip at the beginning of the file

    Returns
    -------
    d : list, list containing contents of file

    """
    with open(file_path, 'r') as f:
        i=0
        d = []
        for l in f.readlines():
            if i >= skp_lin:
                d.append(l.split("\t"))
            i+=1
    return d



def calc_run_count(comp_count, run_data):
    """
    Calculates the number of lateral effect calculations for this batch.

    Parameters
    ----------
    comp_count : int, number of components
    run_data : dict, dictionary of lists of parameters to run calcs for

    Returns
    -------
    rf : int, number of run permutations

    """
    rf = 1
    for k in run_data.keys():
        if k != "states":
            rf *= len(run_data[k])
    rf*=comp_count
    return rf

def read_run_info(run_file):
    """
    Reads set of inputs for which to calculate lateral effects for.

    Parameters
    ----------
    run_file : str, path to properly formatted text file with input parameters



    Returns
    -------
```

```
run_par : dict, all runfile parameters



"""
with open(run_file) as f:
    lines = f.readlines()
## dictionary for raw run data
run_par = {}
## dictionary for unique permuations of run data
runs = {}
## temporary storage for run data
x=[]
## integer key for dictionary of run unique run parameters
i = 0
for l in lines:
    l = l.strip()
    if l[0] != "#":
        p, d = l.split(":")
        d = d.strip().split(",")
        if p == "states":
            x = [s.strip() for s in d]
        else:
            x = [float(f.strip()) for f in d]
        ## ensure only unique values in each parameter set
        run_par[p] = list(set(x))

for ddn_i in run_par['drawdown_init']:
    for ddn_f in run_par['drawdown_final']:
        for d_dia in run_par['drain_diam']:
            for d_dep in run_par['drain_depth']:
                for b_dep in run_par['barrier_depth']:
                    for ss in run_par['surf_storage']:
                        for t in run_par['drawdown_time']:
                            ## enforce physical constraints.
                            ## consider adding ratio of d_dep:b_dep
                if ((0 <= ddn_i < ddn_f < d_dep < b_dep < 15)\
                    and t > 0)\
                    and ((ss == 0 and ddn_i >0) or\
                     (ss >=0 and ddn_i == 0)):
                        runs[i]={'drawdown_init': ddn_i,
                                'drawdown_final': ddn_f,
                                'drain_diam': d_dia,
                                'drain_depth': d_dep,
                                'barrier_depth': b_dep,
                                'surf_storage': ss,
                                'drawdown_time': t}
                        i+=1
                else:
                    print("no go")
    return run_par['states'], runs

def qry_sda(sql):
    """
    requests soil data from sda web service
    For documentation of using web services for SDA see
    https://sdmdataaccess.nrcs.usda.gov/WebServiceHelp.aspx#RunQuery

    Parameters
    ----------
    sql : str, sql string for sda web service

    Returns
    -------
    sda_data : list, returned soil data from sda

    """
    ## sda webservice url
    url = "https://SDMDataAccess.sc.egov.usda.gov/Tabular/post.rest"

    ## data formate
    tab_fmt = "JSON+COLUMNNAME"

    ## parameter dictionary to pass with post request
    params = {"QUERY":sql, "FORMAT":tab_fmt}

    ## execute the post request
    x = req.post(url, data=params)

    ## extract rows of data row from request
    sda_data = x.json()['Table'][1:]

    return sda_data

def qry_rosetta(r_input, r_version):
    """
    request to retrieve rosetta estimates of soil properties
    from Todd Skaggs' (ARS) web-service'
    For documentation of using web services for ROSETTA see
    https://www.handbook60.org/home/

    Parameters
    ----------
    r_input : list, horizon level texture information for ROSETTA

    r_version:, int, rosetta version to use
                    1: original schaap version
                    2: nrcs version
                    3: zhang n schaap 2017
    Returns
    -------
    r : list, returned soil hydraulic properties for all horizons

    """
    rosetta_url = f"http://www.handbook60.org/api/v1/rosetta/{r_version}"
    print("Requesting Rosetta estimates ...")
    r = req.post(rosetta_url, json={"X": r_input})
    if not r.ok:
        print(f"Error!\nStatus code: {r.status_code}\nMessage:\n{r.text}")
        sys.exit()
    r.json()["van_genuchten_params"]
    return r.json()["van_genuchten_params"]


def organic_ptf(dat):
    """
    @author- Jason Roth WME NRCS-MN. jason.roth@usda.gov
    This function populates soil hydraulic parameters necassary for
    calculation of lateral effects in organic soils.

    Saturated hydraulic conductivity, saturated and residual water contents,
    and bulk densities are populated using a lookup table of these values
    based on soil texture. Values were calcualated from analyses of these
    parameters by texture across the PPR.

    PTFs to generate VG hydraulic parameters for organic soils
    adapted from Liu and Lennartz, 2018
    https://onlinelibrary.wiley.com/doi/abs/10.1002/hyp.13314
    All organic PTFs require bulk density

    Future Dev: consider adding refinement based on bulk density as
            shown in original publication

    Parameters
    ----------
    data- numpy array containing soil data

    Returns
    -------
    dat - numpy array containing soil data
    """
```

```
## dictionary to crosswalk database textures to assignments
org_txt_map = {"CE":"MUCK","FLV-":"MPT","FL-":"MPT","HB-M":"MUCK",
            "HPM":"MUCK","MARL":"MUCK","MK-P":"MPT",
            "MK-S":"MUCK", "MPM":"MPT","MPT":"MPT",
            "MUCK":"MUCK","PEAT":"PEAT",
            "SP":"MUCK", "SPM":"PEAT", "SR-":"MUCK",
            "STX-":"MPT", "SR- MUCK FS":"MUCK", "COP-":"MUCK"}

## dictionary to reference hydraulic paramters to assign to ea.
## designated texture class
## Ks in log([cm/d]), RhoB in [g/cm^3], water contents in [cm3/cm3]
org_prams = {"MUCK":{"Ks":2.33, "ThetaS":0.64, "ThetaR":0.17, "RhoB":0.26},
          "MPT":{"Ks":2.90, "ThetaS":0.65, "ThetaR":0.14, "RhoB":0.16},
          "PEAT":{"Ks":3.41, "ThetaS":0.67, "ThetaR":0.09, "RhoB":0.11}
          }

## conversion factor for um/s -> cm/d
ums2cmd = 24.*3600/10**4

## get the names of horizons that we want to populate in this process
org_key = org_txt_map.keys()

## iterate over all horizons
for i in range(dat.shape[0]):
   ## check if the texture is contained in textures to re-assign
   t = dat[i]['texture']
   if t in org_key:
      ## get the re-assignment value
      txt = org_txt_map[t]
      ## calculate average horizon depth to be used in VG n param calc
      d = (dat[i]['hzdepb_r'] + dat[i]['hzdept_r'])/2.

      ## store bulk density in var. for readibility
      if dat[i]['dbthirdbar_r'] < 0:
            dat[i]['dbthirdbar_r'] = org_prams[txt]['RhoB']
        bd = dat[i]['dbthirdbar_r']

        ## assign hydraulic conductivity to this soil
        if dat[i]['ksat_r'] < 0:
            ## NOTE Value in lookup dictionary is already log x-formed
            dat[i]['Ks'] = org_prams[txt]['Ks']
        else:
            ## convert NASIS value  from um/s to cm/d, log xform and store
            dat[i]['Ks'] = math.log10(dat[i]['ksat_r']*ums2cmd)

        ## assign saturated water content to this soil
        if dat[i]['wsatiated_r'] < 0:
            dat[i]['ThetaS'] = org_prams[txt]['ThetaS']
        else:
            dat[i]['ThetaS'] = dat[i]['wsatiated_r']

        ## assign residual water content to this soil
        if dat[i]['wthirdbar_r'] < 0:
            dat[i]['ThetaR'] = org_prams[txt]['ThetaR']
        else:
            dat[i]['ThetaR'] = dat[i]['wthirdbar_r']

        ## Van Genuchten parameters need to be generated for all organics
        ## calculate and store VG log('alpha') parameter for this soil
        ## from Liu and Lennartz, 2018 Table 4
        dat[i]['alpha'] = (0.326-9.315*bd+10.420*bd**2-0.014*d)
        ## calculate and store VG log('n') to this soil
        ## from Liu and Lennartz, 2018 Table 4
        dat[i]['n'] = (0.153-0.422*bd+0.45*bd**2)

    return dat

def calc_soil_conductivity_nrcs(sol_arr, drn_dep, bar_dep, top_dep=0.0):
    """
```

```
calculated hydraulic conductivity above and below drain

assumes soil horizon depths are in cm, hydraulic conductivity log(cm/d)

    Parameters
----------
soll_arr: np recarray, containing depth (cm) and Ks log(cm/d) data for
            a soil component. Array must have named
            fields ['hzdepb_r', 'Ks' ]

drn_dep: float, drain depth (cm)

bar_dep: float, barrier depth (cm)

Returns
-------
ka: float, hydraulic conductivity (cm/d) above drain depth
kb: float, hydraulic conductivity (cm/d) below drain depth

"""

if drn_dep >= bar_dep:
    bar_dep = drn_dep + 1.0

old_bot_dep = top_dep

## state variable for calculating above (1) or below (2) drain
lyr = 1
ksat = 0
lyr_lst = range(sol_arr.shape[0])

sol_arr.sort(order='hzdepb_r')
## figure out what horizon we start in
i = 0
while top_dep > sol_arr['hzdepb_r'][i]:
    i+=1

for i in range(i, sol_arr.shape[0]):
    bot_dep = sol_arr['hzdepb_r'][i]
    k = sol_arr['Ks'][i]

    # determine if new bottom depth is greater than drain depth and
    # need to increment to conductivity below drain and or this is the
    # last depth entry for this component and need to extrapolate values
    # downward to confining layer.

    ## this catches condition where horizon intersects drain depth or
    ## process runs out of soils data and needs to be extrapolated down
    ## to the barrier depth.
    if (bot_dep >= drn_dep and lyr == 1) or i == lyr_lst[-1]:
        ## Both of these ifs can trigger sequentially if drain depth
        ## is below the bottom depth of the deepest soil horizon
        ## case - horizon intersects drain depth
        if lyr == 1:
            ## incremental layer thickness
            thk = (drn_dep - old_bot_dep)
            ## calculate total layer thickness down to drain depth
            lyr_thk = drn_dep
            ## undo log-xform and multiply by incremental thickness and
            ## add to the running sum of products
            ## std_dev = k_u = 0
            ksat += thk * 10**(k)
            ## divide total sum product by total layer thickness to get
            ## depth weighted average of horizontal conductivity
            ka = ksat/lyr_thk
            ## increment the layer counter and calculate for first
            ## increment below drain and
            lyr = 2
            ## incremental layer thickness
            thk = (bot_dep - drn_dep)
```

```
                ## undo log-xform and multiply by incremental thickness and
                ## add to the running sum of products
                ## std_dev = k_u = 0
                ksat = thk * 10**(k)
                old_bot_dep = bot_dep
            ## case - ran out of soils data
            if i == lyr_lst[-1]:
                ## check if last soils increment is less than the barrier depth
                ## in this case we need to extrapolate downward to the
                ## barrier depth. There has been discussion of using a decay
                ## factor based on the assumption that conductivity is reduced
                ## at depth due to consolidation
                if bot_dep < bar_dep:
                    bot_dep = bar_dep
                ksat += (bot_dep - old_bot_dep)*10**(k)
                lyr_thk = (bot_dep - drn_dep)
                kb = ksat/lyr_thk
                old_bot_dep = 0
        else:
            thk = (bot_dep - old_bot_dep)
            ## undo log-xform and multiply by incremental thickness

            ksat += thk * 10**(k)
            old_bot_dep = bot_dep

    return ka, kb



def calc_soil_porosity_nrcs(sol_arr, wtb_ini=0.0, wtb_fin=1.0, inc=1.0):
    """
    NRCS-EFH Ch 19-60
    calculate drainable porosity as function of head between 2 drawdown depths
    i.e. water retention drawdown curve using Van Genuchten relationships.

    Parameters
```

```
----------
soll_arr: np recarray, containing depth (cm) and Ks log(cm/d) data for
        a soil component. Array must have named
        fields ['hzdepb_r', 'ThetaS', 'ThetaR', 'alpha', 'n' ]

wtb_ini: float, initial water table depth (cm)

wtb_fin: float, final water table depth (cm)

Returns
-------
f: float, drainable porosity (in/in) between drawdown depths

"""

wtb = wtb_ini

hed = 0

## initialize drainable porosity at 0
f = 0
## set soil horizon index variable to 0
a = 0
## ensure soil values are sorted.
sol_arr.sort(order='hzdepb_r')

## make sure calculation begins in horizon containing the intitial
## water table depth
while sol_arr['hzdepb_r'][a] < wtb_ini and a < sol_arr.shape[0]-1:
  a += 1
## set initial temp value of bot to get into loop...sloppy.
bot = 1.0

## iterate over depth btwn initial and final water table in 1cm increments
while wtb <= wtb_fin:
    if (wtb > bot and a < sol_arr.shape[0]-1) or wtb==wtb_ini:
        ## if this depth is greater than the horizon depth increment "a"
        ## so functions reference corect soil parameters for this depth
        if a < sol_arr.shape[0]-1 and wtb >= bot:
            a += 1
        ## populate parameters for VG equation for this horizon
        bot = min(sol_arr['hzdepb_r'][a], wtb_fin)
        ths = sol_arr['ThetaS'][a]
        thr = sol_arr['ThetaR'][a]
        enn = sol_arr['n'][a]
        alfa = sol_arr['alpha'][a]
        ## tranform from log values
        enn = 10**enn
        alfa = 10**alfa
        emm = 1.-1./enn
    ## calculate total water drained to this depth
    ## theta_s minus VG water retention at this depth : ch 19, eqn 19-4
    if wtb + inc < wtb_fin:
        hed = wtb_fin - (wtb+inc/2.)
    else:
        hed = (wtb_fin - wtb)/2.

    if (ths - (thr + (ths - thr)) <= 0 or \
        (1.0 + abs(alfa*hed)**enn)**emm) <= 0:
        print("halt")

    f += ths - (thr + (ths - thr) /
            (1.0 + abs(alfa*hed)**enn)**emm)

    wtb+=inc
## normalize drainable porosity to depth drained to get average
## cm/cm drained for the drawdown depth
f/=(wtb_fin-wtb_ini)
```

```
    # if drainable porosity is greater than 0.5 this method is not appropriate
    if f > 0.5:
        f = -9.9
    if np.isnan(f):
        f = -9.9

    return f


def calc_vansch(wtb_ini, wtb_fin, drn_dep, drn_dia, bar_dep,
                hcn_abv, hcn_bel, drn_por, sur_str, tim_per):
    """
    DESC : calculate a lateral effect using the VanS method for a specified
    time period. adapted from:
    https://www.nrcs.usda.gov/Internet/FSE_DOCUMENTS/nrcs142p2_024926.pdf
    Parameters
    ----------
    wtb_ini: float, initial water table depth below grade (ft)

    wtb_fin: float, final water table depth below grade (ft)

    drn_dep: float, drain depth below grade (ft)

    drn_dia: float, drain diam (in)

    bar_dep: float, depth below grade to impervious barrier (ft)

    hcn_abv: float, hydraulic conductivity above drain (ft/d)

    hcn_bel: float, hydraulic conductivity below drain (ft/d)

    drn_por: float, drainable porosity (-/-)

    sur_str: float, surface storage of water (ft)

    tim_per: float, hydraulic conductivity above drain (ft/d)
```

```
Returns
-------
le: float, lateral effect, half the drain spacing that results in specified
        drawdown over the specified time period (ft).
"""
## assign some constants
pie = 3.1416
tol = 0.01
eff_dep = bar_dep - drn_dep
## get effective radius from function
eff_rad = get_eff_rad(drn_dia)
## set some interim variables
aye = bar_dep - drn_dep
emm_ini = drn_dep - wtb_ini
emm_fin = drn_dep - wtb_fin
eff_por = drn_por + (sur_str /(12.*(emm_ini - emm_fin)))
## calculate representative hcon
eff_hcn = ((hcn_abv * emm_ini) + (hcn_bel * eff_dep)) / (emm_ini + eff_dep)
## get initial ess_pri using the actual depth of drain
num = 9. * eff_hcn * tim_per * eff_dep
den = eff_por * (math.log(emm_ini * (2. * aye + emm_fin)) -
            math.log(emm_fin * (2. * aye + emm_ini)))
ess = 1000
if num > 0 and den > 0:
    ess_pri = math.sqrt(num / den)
    ## initialize 'err'>'tol' so we commit to while loop atleast 1x
    err = tol * 1.1
    ## initialize 'err'>'tol' so we commit to while loop atleast 1x
    ess = 0
    ## loop while error is greater than
    ## specified tolerance (NRCS recommends 10% tolerance)
    while (err > tol):
        ## determine eff_dep
        if aye / ess_pri <= 0.3:
            eff_dep = aye / (1. + (aye / ess_pri) *
                            ((8. / pie) * math.log(aye / eff_rad) - 3.4))
        else:
            eff_dep = ess_pri * pie /\
                        (8. * (math.log(ess_pri / eff_rad) - 1.15))
        ## calculate new spacing for this iteration
        num = 9. * eff_hcn * tim_per * eff_dep
        den = eff_por * (math.log(emm_ini * (2. * eff_dep + emm_fin)) -
                         math.log(emm_fin * (2. * eff_dep + emm_ini)))
        if num > 0 and den > 0:
            ess_pri = math.sqrt(num / den)
        else:
            print(drn_por, hcn_abv, hcn_bel)
        err = abs(ess - ess_pri) / ess_pri
        ess = ess_pri
    return ess / 2.0

def get_eff_rad(in_dia):
    """
    DESC: simple lookup of effective radius for a drain based on diameter

    Parameters
    ----------
    in_dia : int, diameter of drain

    Returns
    -------
    Re : float, effective radius of drain

    """
    if in_dia <= 3:
        Re = 0.0115
    elif in_dia > 3 and in_dia <= 4:
        Re = 0.0167
    elif in_dia > 4 and in_dia <= 5:
        Re = 0.034
    elif in_dia > 5 and in_dia <= 6:
        Re = 0.048
    elif in_dia > 6 and in_dia <= 8:
        Re = 0.08
    elif in_dia > 8 and in_dia <= 10:
        Re = 0.111
    elif in_dia > 10 and in_dia <= 12:
        Re = 0.142
    elif in_dia > 12 and in_dia <= 16:
        Re = 0.25
    else:
        Re = 1.0
    return Re

def check_soils(ptf_data, min_pct=15, maxmin_dep=10., minmax_dep=120.,
                nul_val=-9.9):
    """
    run through soils and make sure that theres enough data and if
    not explain how

    Parameters
    ----------
    ptf_data : numpy recarray, processed soils data needed to calculated
                            lateral effects. Expects

    min_pct : int, min component percent for mapunit to be valid

    maxmin_dep : int, maximum minimum depth at which a horizon data are needed
                    to produce a value for this component

    maxmin_dep : int, minimum maximum depth at which a horizon data are needed
                    to produce a value for this component

    nul_val : float, the value for missing data
```

```
Returns
-------
mky_chk : numpy recarray, mukeys indicating whether enough data
          is present for lateral effect determination

cky_chk : numpy recarray, cokeys indicating whether enough data
          is present for lateral effect determination
"""

mky_type = ['U6', 'U8', 'U8', 'i4']

mky_cols = ['asym', 'mukey', 'musym', 'code']

mky_dt = list(zip(mky_cols, mky_type))

cky_type = ['U6', 'U8', 'U8', 'U8', 'U20', 'i4']

cky_cols = ['asym', 'mukey', 'musym', 'cokey', 'compname', 'code']

cky_dt = list(zip(cky_cols, cky_type))

## container for mapunit designations based on analysis
mky_lst = []

cky_lst = []

## get unique mukeys to iterate over
mukeys = np.unique(ptf_data['mukey'])

## get rid of all null data
ptf_data = ptf_data[np.where(
                    np.logical_and(ptf_data['Ks'] != nul_val,
                    np.logical_and(ptf_data['ThetaR'] != nul_val,
                    np.logical_and(ptf_data['ThetaS'] != nul_val,
                    np.logical_and(ptf_data['alpha'] != nul_val,
                    ptf_data['n'] != nul_val)))))]


## now compare components for each mukey
for mkey in mukeys:
    pct = 0
    # get unique components for this map unit
    mkey_data = ptf_data[np.where(ptf_data['mukey']==mkey)]

    if mkey_data.shape[0] > 0:
        asym = mkey_data['areasymbol'][0]
        msym = mkey_data["musym"][0]
        mkey = mkey_data["mukey"][0]
        ## is there any data for the mapunit

        ## iterate over unique components in mapunts and store max result
        cokeys = np.unique(mkey_data['cokey'])
        for ckey in cokeys:
            ## store component soil data
            comp_data = mkey_data[mkey_data['cokey']==ckey]
            ## is there any data for this component
            if comp_data.shape[0] >= 0:
                cnam = comp_data['compname'][0]
                ckey = comp_data["cokey"][0]
                cpct = comp_data["comppct_r"][0]
                ## sort existing component data by depth
                ## determine if there's top and bottom soil data
                #print("max depth {0}, {1}".format(
                #                           comp_data['hzdepb_r'].max(),
                #                            cpct))
                if comp_data['hzdept_r'].min() <= maxmin_dep and \
                    comp_data['hzdepb_r'].max() > minmax_dep:
                    pct += cpct
                else:
```

```
                cky_lst.append([asym, mkey, msym, ckey, cnam, -2])
            else:
                cky_lst.append([asym, mkey, msym, ckey, cnam, -1])

        ## is there enough of all components to satisfy a lateral
        ## effect calculation for this mapunit
        if pct < min_pct:
            mky_lst.append([asym, mkey, msym, 0])
        else:
            mky_lst.append([asym, mkey, msym, 1])
    else:
        mky_lst.append([asym, mkey, msym, 0])

    cky_chk = np.array([tuple(c) for c in cky_lst], dtype=cky_dt)
    mky_chk = np.array([tuple(m) for m in mky_lst], dtype=mky_dt)
    return mky_chk, cky_chk


def calc_lateral_effects(ptf_data, run_data):
    """
    DESC: calculates lateral effect for unique soil mapunits
    contained in ptf_data using parameters from run_data and stores the
    results in led_data.

    Parameters
    ----------
    ptf_data : numpy recarray
        processed soils data needed to calculated lateral effects. Expects
        column headers of
    led_data : numpy recarray
        calculated lateral effect distance for each mapunit/run parameter
        permutation
    run_data : Dict
        Dictionary containing all permutations of unique input parameters.
    Returns
```

```
-------
led_data : list
    calculated lateral effect distance for each mapunit/run parameter
    permutation
    [['areasymbol', 'mukey', 'musym', 'cokey', 'compname', 'comppct',
            'drain_depth', 'drain_diam', "surf_storage", "barrier_depth",
            "drawdown_init", "drawdown_final", "drawdown_time",
            "Kh_abv", "Kh_bel", "eff_por", "leff_dist"]]

no_led_data : list
    counties, mapunits and components that didn't have sufficient data for
    lateral effect calculations
     [['areasymbol', 'mukey', 'musym', 'cokey', 'compname', 'comppct']]
"""
i = 0

ft2cm = 12.0*2.54
min_pct = 50.0
## get number of unique components
rukeys = run_data.keys()
pct_inc = 5
rc = 0
rpct = 5
prog_msg = "{0}% done with calculations for {1}"
st = ptf_data['areasymbol'][0][0:2]

null_mkey = []
led_data = []

### check soils for complete data
mky_chk, cky_chk = check_soils(ptf_data, min_pct=50,
                    maxmin_dep=10.,
                    minmax_dep=120.,
                    nul_val=-9.9)
```

```
mukeys = np.unique(ptf_data['mukey'])

rcnt = mukeys.shape[0]*len(rukeys)
print("calculating lateral effects for {0}".format(st))
print("total run count is {0}".format(rcnt))


## determine which components for this mapunit
## have sufficient data for LE cal
for r in rukeys:
    rd = run_data[r]
    ddn_i = rd['drawdown_init']
    ddn_f = rd['drawdown_final']
    d_dia = rd['drain_diam']
    d_dep = rd['drain_depth']
    b_dep = rd['barrier_depth']
    ss = rd['surf_storage']
    t = rd['drawdown_time']

    for mk in mukeys:
        code = mky_chk[np.where(mky_chk['mukey']==mk)]['code'][0]
        rc += 1
        if rc/rcnt*100.0 > rpct:
            print(prog_msg.format(rpct, st))
            print(rc, rcnt)
            rpct+=pct_inc
        mu_data = ptf_data[np.where(ptf_data['mukey']==mk)]
        asym = mu_data['areasymbol'][0]
        msym = mu_data["musym"][0]
        mkey = mu_data["mukey"][0]
        if code > 0:
            ## set initial lateral effect for this mapunit to zero
            max_now = 0
            ## iterate over unique map units in data
            ## make sure this is sorted
```

```
tot_pct = 0
cokeys = np.unique(mu_data['cokey'])
for ck in cokeys:
    comp_data = ptf_data[np.where(ptf_data['cokey']==ck)]
    comp_data.sort(order="hzdept_r")
    cnam = comp_data['compname'][0]
    ckey = comp_data["cokey"][0]
    cpct = comp_data["comppct_r"][0]
    ## convert values to cm in f(x) call
    f = calc_soil_porosity_nrcs(comp_data, ddn_i*ft2cm,
                               ddn_f*ft2cm, inc=1.0)
    ## functions expects values in cm
    hka, hkb = calc_soil_conductivity_nrcs(comp_data,
                                          d_dep*ft2cm,
                                          b_dep*ft2cm)
    ## convert values to ft/d
    hka/=ft2cm
    hkb/=ft2cm
    ## final check on data and need for run
    if f > 0.0 and hka > 0.0 and hkb > 0.0:
    ## function expects values in ft and days.
    ## assume constant barrier depth and
    ## initial water table for now
        le = calc_vansch(ddn_i, ddn_f, d_dep, d_dia,
                         b_dep, hka, hkb, f, ss, t)

        le = (int(le/10.)+1)*10.
        tot_pct += cpct
        ## check if this lateral effect is larger than
        ## the previous max for this mapunit
        if le > max_now:
            max_now = le

if tot_pct >= min_pct:
```

```
                dat = [asym, mkey, msym, ckey, cnam, cpct,
                       d_dep, d_dia, ss, b_dep, ddn_i, ddn_f,
                       t, hka, hkb, f, max_now]

            else:
                dat = [asym, mkey, msym,  99999999, "unknown", 0,
                       d_dep, d_dia, ss, b_dep, ddn_i, ddn_f,
                       t, -9.9, -9.9, -9.9, -2]

        else:
            dat = [asym, mkey, msym,  99999999, "unknown", 0,
                   d_dep, d_dia, ss, b_dep, ddn_i, ddn_f,
                   t, -9.9, -9.9, -9.9, -1]

        led_data.append(dat)
        i+=1

    print("finsished calculating lateral effects for {0}".format(st))
    return led_data, null_mkey

### MAIN FUNCTION
def main():
    """
    DESC: main control loop

    Parameters
    ----------
    none

    Returns
    -------
    none

    """
```

```
run_file_name = "input.txt"
output_dir = "leff_data"

## formattable strings for state specific data files
sda_file = "{0}_sda_qry.txt"
ptf_file = "{0}_rout.txt"
led_file = "{0}_led.txt"
no_led_file = "{0}_no_led.txt"

delim="\t"

## query text
sql_compsWithData = '''
SELECT legend.areasymbol, mapunit.mukey, mapunit.musym,
component.cokey, component.compname,component.comppct_r,
chorizon.chkey, chtexturegrp.texture,
ISNULL(chorizon.ksat_r, -9.9) as 'ksat_r',
ISNULL(chorizon.wsatiated_r, -9.9)/100.0 as 'wsatiated_r',
chorizon.hzdept_r, chorizon.hzdepb_r,
ISNULL(chorizon.sandtotal_r, -9.9) as 'sandtotal_r',
ISNULL(chorizon.silttotal_r, -9.9) as 'silttotal_r',
ISNULL(chorizon.claytotal_r, -9.9) as 'claytotal_r',
ISNULL(chorizon.dbthirdbar_r, -9.9) as 'dbthirdbar_r',
ISNULL(chorizon.wthirdbar_r, -9.9)/100.0 as 'wthirdbar_r',
ISNULL(chorizon.wfifteenbar_r, -9.9)/100.0 as 'wfifteenbar_r'
FROM (legend LEFT JOIN
        (mapunit LEFT JOIN
        (component LEFT JOIN
        (chorizon LEFT JOIN chtexturegrp ON chorizon.chkey = chtexturegrp.chkey)
        ON component.cokey = chorizon.cokey)
        ON mapunit.mukey = component.mukey)
        ON legend.lkey = mapunit.lkey)
WHERE legend.areasymbol like '{0}%' AND component.comppct_r>=10
AND chtexturegrp.rvindicator = 'Yes' AND mapunit.musym <> 'W'
AND (chorizon.hzname NOT LIKE 'Cr%')
```

```
ORDER BY mapunit.mukey, component.cokey, chorizon.hzdepb_r
'''


sql_allComps = '''
    SELECT legend.areasymbol, mapunit.mukey, mapunit.musym,
    component.cokey, component.compname,component.comppct_r,
    chorizon.chkey
    FROM (legend LEFT JOIN
                  (mapunit LEFT JOIN
                  (component LEFT JOIN chorizon ON component.cokey = chorizon.cokey)
                  ON mapunit.mukey = component.mukey)
                  ON legend.lkey = mapunit.lkey)
    WHERE legend.areasymbol like '{0}%' AND component.comppct_r>=10
    AND mapunit.musym <> 'W'
    ORDER BY mapunit.mukey, component.cokey
    '''


## header and dataype for sda query results
sda_data_cols = ['areasymbol', 'mukey', 'musym', 'cokey', 'compname',
         'comppct_r', 'chkey', 'texture', 'ksat_r', 'wsatiated_r',
         'hzdept_r', 'hzdepb_r', 'sandtotal_r', 'silttotal_r',
         'claytotal_r', 'dbthirdbar_r','wthirdbar_r', 'wfifteenbar_r']

sda_data_type = ["U8", "U8", "U10", "U8", "U32",
         "i2", "U8", "U8", "f4", "f4",
         "i2", "i2", "f4", "f4",
         "f4", "f4", "f4", "f4"]

sda_data_unit = ["sta_fip", "key", "name", "key", "name",
         "%", "key", "txt", "um/s", "cm3/cm3",
         "cm", "cm", "%", "%",
         "%", "g/cm3", "cm3/cm3", "cm3/cm3"]

sda_hdr = make_file_hdr(sda_data_cols.copy(), sda_data_unit.copy(), delim)
```

```
sda_comp_cols = sda_data_cols[0:6]
sda_comp_type = sda_data_cols[0:6]

## this is the order that data will come back
ptf_cols = ['areasymbol', 'mukey', 'musym', 'cokey', 'compname',
        'comppct_r', 'chkey', 'texture', 'hzdept_r', 'hzdepb_r',
        "ThetaR", "ThetaS", "alpha", "n", "Ks"]

ptf_type = ["U8", "U8", "U10", "U8", "U32",
        "i2", "U8", "U8", "i2", "i2",
        "f4", "f4", "f4", "f4", "f4"]

ptf_unit = ["sta_fip", "key", "name", "key", "name",
        "%", "key", "txt", "cm", "cm",
        "cm3/cm3", "cm3/cm3", "log(cm/d)", "log(1/cm)", "log(cm/d)"]

ptf_hdr = make_file_hdr(ptf_cols.copy(), ptf_unit.copy(), delim)

led_cols = ['areasymbol', 'mukey', 'musym', 'cokey', 'compname', 'comppct',
         'drain_depth', 'drain_diam', "surf_storage", "barrier_depth",
         "drawdown_init", "drawdown_final", "drawdown_time",
         "Kh_abv", "Kh_bel", "eff_por", "leff_dist"]

led_type = ["U8", "U8", "U8", "U8", "U32", "i2",
         "f4", "f4", "f4", "f4",
         "f4", "f4", "f4",
         "f4", "f4", "f4","i4"]

led_unit = ["sta_fip", "key", "name", "key", "name", "%",
        "ft", "in", "in", "ft",
        "ft", "ft", "d",
        "ft/d", "ft/d", "ft/ft", "ft"]

led_hdr = make_file_hdr(led_cols.copy(), led_unit.copy(), delim)
```

```
## make a subdirectory to store data in
data_dir = os.path.join(os.getcwd(), output_dir)
if not os.path.exists(data_dir):
    os.mkdir(data_dir)

run_file_path = os.path.join(os.getcwd(), run_file_name)

if os.path.exists(run_file_path):
    states, run_data = read_run_info(run_file_path)

## iterate over each state in the list of states specified in run_file
## tried querying more than one state at a time but POST request fails
for s in states:
    run_type = 0
    ## determine if any of the data exist for this state
    if os.path.exists(os.path.join(data_dir, led_file.format(s))):
        run_type = 3
    elif os.path.exists(os.path.join(data_dir, ptf_file.format(s))):
        run_type = 2
    elif os.path.exists(os.path.join(data_dir, sda_file.format(s))):
        run_type = 1
    else:
        run_type = 0

    ## if run type < 3 some work needs to be done.
    if run_type < 3:
        ## if there is no soil query in data directory need to query
        ## soil data for this state from SDA
        if run_type <= 1:
            ## if soil data already exists import it to run through rosetta
            sda_file_pth = os.path.join(data_dir,sda_file.format(s))
            sda_dt = list(zip(sda_data_cols,sda_data_type))
            if run_type == 1:
                print("Local SDA data found for {0}".format(s))
                print("Importing soil data from " + sda_file.format(s))
                sda_data = read_data_file(sda_file_pth, skp_lin=2)
                for i in range(len(sda_data)):
                    for j in range(8,18):
                        sda_data[i][j] = float(sda_data[i][j])

            else:
                # query SDA using web service
                print("Soil data not found for {0}".format(s))
                print("Querying soil data from SDA")
                sda_data = qry_sda(sql_compsWithData.format(s))

                ## construct a record array from the query results.
                ## rec arrays are much quicker in execution than dataframes
                ## need to convert to tuple so sequential fields of same
                ## data type aren't combined
                arr = np.array([tuple(sd) for sd in sda_data],
                            dtype=sda_dt)

                ## replace all values in ksat_r, wsatiated_r, sandtotal_r
                ## silttotal_r, claytotal_r, dbthirdbar_r, wthirdbar_r
                ## wfifteenbar_r

                for f in ["ksat_r", "wsatiated_r", "sandtotal_r",
                        "silttotal_r", "claytotal_r", "dbthirdbar_r",
                        "wthirdbar_r", "wfifteenbar_r"]:

                    arr[f][np.where(arr[f]<0.0)] = -9.9000



                ## add placeholder entries for soil map units that
                ## didn't return any data from SDA
                ## get all the comps in the state and compare them to
                ## to those with data and fill in the comps that don't
```

```
        ## have data with null
        print("Querying component data from SDA")
        sda_comp_data = qry_sda(sql_allComps.format(s))
        #comp_dt = list(zip(sda_comp_cols,sda_comp_type))

        no_hzn = []

        sda_comp_data = qry_sda(sql_allComps.format(s))

        ## make
        for scd in sda_comp_data:
            if scd[-1] == None:
                x = scd[0:6]+['99999999']+['UKN']+2*[-9.9] +\
                    [0, 10] + 6*[-9.9]
                no_hzn.append(x)
                sda_data.append(x)

        cmp_arr = np.array([tuple(cd) for cd in no_hzn],
                       dtype=sda_dt)

        arr = np.append(arr, cmp_arr)

        ## save a copy of the sda query to a file for future ref
        np.savetxt(sda_file_pth, arr , fmt="%s", delimiter=delim,
                   header=sda_hdr, comments="")
        del arr
if run_type <= 2:
    ## soils data for this state has already been processed
    ptf_file_pth = os.path.join(data_dir,ptf_file.format(s))
    if run_type == 2:
        print("Local PTF data found for {0}".format(s))
        print("Importing PTF data from " +ptf_file.format(s))
        ptf_data = read_data_file(ptf_file_pth, skp_lin=2)
        dt = list(zip(ptf_cols, ptf_type))
        ptf_data = [tuple(i) for i in ptf_data]
    ptf_data = np.array(ptf_data, dtype=dt)
else:
    print("PTFs data not found for {0}".format(s))
    print("Processing soils through PTFs")
    ptf_data = qry_rosetta([sd[12:] for sd in sda_data], 2)

    ## need to convert to tuple so sequential fields of same
    ## data type aren't combined

    ## merge sda_data and rosetta output
    temp = []
    for i,j in zip(sda_data, ptf_data):
        temp.append(tuple(i+j))
    dt = list(zip(sda_data_cols+ptf_cols[-5:],
                  sda_data_type+ptf_type[-5:]))
    temp = np.array(temp, dtype=dt)
    temp= organic_ptf(temp)

    ptf_data = temp
    del temp

    ptf_data['Ks'][np.isnan(ptf_data['Ks'])] = -9.9
    ptf_data['ThetaR'][np.isnan(ptf_data['ThetaR'])] = -9.9
    ptf_data['ThetaS'][np.isnan(ptf_data['ThetaS'])] = -9.9
    ptf_data['alpha'][np.isnan(ptf_data['alpha'])] = -9.9
    ptf_data['n'][np.isnan(ptf_data['n'])] = -9.9



    np.savetxt(ptf_file_pth,
               ptf_data[[c for c in ptf_cols]],
               fmt="%s",
               delimiter=delim,
               header=ptf_hdr,
               comments="")
        if run_type <= 3:
            led_file_pth = os.path.join(data_dir,led_file.format(s))
            no_led_file_pth = os.path.join(data_dir,no_led_file.format(s))
            led_dt = list(zip(led_cols,led_type))

            #no_led_file_pth = os.path.join(data_dir, no_led_file.format(s))
            if run_type == 3:
                print("Lateral Effect Distances for {0} exist".format(s))
            else:
                print("Lateral Effect not found for {0}".format(s))
                print("Processing Lateral Effects")

                ## calculate lateral effects
                led_data, mky_chk = calc_lateral_effects(ptf_data,
                                                         run_data)

                arr = np.array([tuple(le) for le in led_data],dtype=led_dt)

                arr.sort(order=['surf_storage','mukey','drain_depth',
                                'drain_diam'])
                np.savetxt(led_file_pth, arr, fmt="%s",
                           delimiter=delim, header=led_hdr, comments="")

                np.savetxt(no_led_file_pth,
                           mky_chk[np.where(mky_chk['code']<=0)],fmt="%s",
                           delimiter=delim, comments="")

if __name__ == "__main__":
    main()
```