# APPENDIX Q
# USLE TECHNICAL DOCUMENTATION

# 1 Conceptual Framework

The RUSLE (Revised Universal Soil Loss Equation) model describes the potential for sheet flow erosion on disturbed land surfaces based on properties of the land slope, local rainfall, soil characteristics, and land management practices. According to the USDA it is land-use agnostic (USDA, 2024) and is well-suited to model disturbance from construction sites.

RUSLE is based on a factored equation of several variables of the form:

$$A = R \cdot K \cdot L \cdot S \cdot C \cdot P$$

Where $A$ is soil loss, $R$ is a rainfall-runoff erosivity factor, $K$ is a soil erodibility factor, $L$ is a slope length factor, $S$ is slope steepness, $C$ is a coverage management factor, and $P$ is a supporting practices factor (Renard et al., 1991). It is an empirical model which captures the first-order effects of sheet flow and rill erosion for disturbed landscapes (Renard et al., 1991). The model was developed by the Department of Agriculture and Soil Conservation Service (Renard et al., 1991).

The model is used widely in regulatory practice, including for DNR stormwater regulation and federal management.

# 2 Application

DNR's existing USLE implementation of USLE takes the form of a macro-enabled excel worksheet suitable for evaluation of single construction projects or relatively compact project areas. The project uses a county-specific rainfall erosivity ($R$) factor and categorical information about soil texture, as well as manually-inputted slope and slope length information and land cover/control practices to determine estimated sediment loss.

The model relies on a series of empirical equations used depending on user-specified site conditions and erosion control practices. The model chooses an empirical relationship or set of equations based on categorical bins from the data to produce locally accurate sediment models.

While this implementation of the USLE model is a relatively straightforward way to handle small-scale erosion estimation, the mode of interaction scales poorly to large projects like the Enbridge Line 5 Reroute. The reimplemented version applies the logical assessment steps from each factor to a spatial extent, specifically per-pixel across a raster surface.

the original model can be defined mathematically as:

$$\Upsilon_s = (R \cdot R_a) \cdot K \cdot S \cdot L_s \cdot C \cdot D_s \cdot C_f$$

$$\Upsilon_{\text{tot}} = \sum_{i}^{i=n} \Upsilon_{s,i}$$

for phases $i \to n$, where $R_i$ is the proportion of runoff falling within the time period $i$, $R_a$ is the annual runoff coefficient, $K$ is the soil erodibility factor, $S$ is the slope percentage, $L_s$ is slope length, $C_i$ is the land cover coefficient at phase $i$, $D_s$ is the sediment discharge factor, and $C_{f_i}$ is the sediment control factor at phase $i$.

If each coefficient is defined in a spatially explicit cell-series, such that parameter $\boldsymbol{P}$ is defined as a matrix where rows $i$ and columns $j$ correspond to X and Y values in space:

$$
\boldsymbol{P} = \begin{pmatrix} P_{1,1} & \cdots & \cdots & {}_{(1,j)}P \\ \vdots & \vdots & \ddots & \vdots \\ P_{i,1} & \cdots & \cdots & P_{i,j} \end{pmatrix}
$$

Then the corresponding surface for a step of soil loss is:

$$
\boldsymbol{\Upsilon} = \boldsymbol{R} \odot \boldsymbol{R_a} \odot \boldsymbol{K} \odot \boldsymbol{S} \odot \boldsymbol{L_s} \odot \boldsymbol{C} \odot \boldsymbol{D_s} \odot \boldsymbol{C_f}
$$

The Hadamard product of the corresponding surfaces. The surface $\boldsymbol{\Upsilon}$ then corresponds to a soil loss estimate per pixel $\boldsymbol{\Upsilon}_{i,j}$.

# 3 Assumptions

In its present implementation, all spatially-explicit parameters are time-invariant. Spatiotemporal evaluation is not impossible but is avoided in the present context for simplicity. Time-varying parameters (for example land cover) are assumed to be spatially uniform.

This limitation is purely for simplicity and could be overcome at the expense of more data processing, and some modification of the equation specification.

Another key assumption of the model is that the slope length is the same as the cell size; i.e. each cell is considered an individual slope. This modifies the behavior of the equation somewhat when assessing spatial soil loss because of the logic associated with different slope lengths. This assumption was made for processing expediency, because the classification of slopes and their lengths is a nontrivial exercise when identifying landscape features and would have added substantially to the processing cost of performing associated calculations.

# 4 Processing Pipeline

The USLE equation was practically implemented for spatial analysis using ArcGIS Pro and its python package ArcPy. The ArcPy script handles computation of the equation values and the logic leading to them. It takes a series of raster and parameter inputs and converts them to raster output representing the soil loss in tons per acre for a spatial area.

## 4.1 Data Sources

Elevation data were sourced from the State Cartographer's Office (SCO, 2019). DEMs are LiDAR scans in Ashland, Bayfield, and Iron counties. LiDAR data form the basis of all raster derivatives and datasets for the analysis. The slope raster used for analysis is derived from the LiDAR data and has the same resolution (10m). The analysis uses a K-Factor raster derived from queries of the Soil Survey Geographic Database (SSURGO) at a depth of 9 to 15 inches (Soil Survey Staff, 2024). The raw map units were converted from polygons to rasters and upsampled to reach 10m resolution during processing. The pipeline workspace footprint is based on privileged data provided to DNR during analysis for the Environmental Impact Statement; this footprint (proivded as a `.shp` file) was used to clip all raster derivatives and make contributing area polygons and even divisions for sediment discharge estimation. Crossings identified in Table 1 are from Enbridge's crossing table, which provides information about each crossing identified by Enbridge survey staff. Stream polygons used to find areas within 50 horizontal meters of a crossing (see Section 4.3.1) were digitized by DNR staff to add to existing stream network products previously created by the Department.

## 4.2 ArcPy Script

The ArcPy script takes slope, k factor, and soil texture rasters as input, and requires input for start and end days for a scenario, a land cover condition (e.g. bare soil, mulch), and an erosion control condition. It composes a series of functions, with the top level function `sediment_assessment()` as the main entry point and modeling interface. `sediment_assessment()` calculates one timestep worth of erosion estimation at a time; that is to say, it integrates erosion for a series of fixed parameters and one defined period of cumulative rainfall. To vary the other parameters, multiple calls of `sediment_assessment()` must be made in series to describe the erosion impacts of those sections separately, then aggregated together into one layer at the end.

### 4.2.1 Evaluating landslope factor:

The LS factor evaluation is a conditional fucntion which mathces the slope length and steepness to a regression equation matching its range and outputting the ls factor. The relations are empirical so the fit from the numbers is more or less exact.

### 4.2.2 Evaluating the primary soil loss factor

`evaluate_soil_loss_a()` takes the start and end date, a k-factor raster, slope raster, and a default k estimate.

The start and end date are used to lookup the cumulative precipitation time series for the tool; the tool is hardcoded to ashland county's assumed average cumulative rainfall but could be easily extended to look up other county numbers and get a different cumulative rainfall total from them.

THe primary factor is the raw value of sediment loss for the area in question, without factoring in ground cover or other values that might modify it somewhat.

### 4.2.3 SDF Factor Evaluation

SDF is the factor which evaluates the actual amount of sediment discharged from the land surface based on soil characteristics.

Each cell in the slope raster goes through a logic tree at the beginning of the function to assign categorical numbers for sand, silt, and clay categories. Then, based on these logical categories, the sediment discharge factor is calculated using an empirical equation matching the categorical numbers evaluated in the first branching section.

### 4.2.4 Final sediment evaluation

Final sediment discharge is evaluated incorporating sediment control practices into the erosion evaluation. The erosion practices remove a flat percentage of the total erosion loss based on performance evaluations of stormwater staff.

### 4.2.5 Wrapper function

All functions are wrapped within a wrapper called `sediment_assessment()`, which combines the logic of each of the steps into one easy package which takes in all the assumptions and outputs a neat final answer without having to write multiple lines every time an evaluation step is necessary.

### 4.2.6 Model Scenario Runner

The function `run_model_scenario()` is a wrapper that takes an iterator of specifications for model inputs and automatically runs and saves a series of model outputs.

## 4.3 Postprocessing in ArcGIS Pro

Each modeling scenario is output as a `.tif` file into a directory of the user's choosing, after which point each output can be postprocessed. Each phase of the construction is first imported into ArcPro, then is run through an automated postprocessing routine to create final aggregated clip regions. The aggregation routine begins with the continuous erosion surfaces generated by the USLE tool (usually for the whole landscape at a given resolution) and a series of areas to which the outputs are clipped and/or aggregated. This allows flexibility in the representation of outputs based on the user's desired aggregation behavior. Model outputs for both modeling scenarios were clipped to two different spatial aggregation schemes.

After clipping, the model outputs were aggregated based on the ModelBuilder routine shown in Figure 1. The processing routine first takes the stages of the model as inputs, then converts their rate numbers to scalar quantities to add together. The converted rasters are summed and reconverted to rates to get the total erosion for the scenario, which is the primary output. Each of the five rasters is then clipped to the contributing area polygons. Then all rasters are summed by zone, aggregating the smaller estimates of erosion into one larger estimate of the erosion for the entire area. This estimate is then converted back to a rate by dividing out the area of the contributing area in question. Because each contributing area is associated with a unique ID, the outputs can then be reported as both layers and as excel files which can later be summarized in other ways. Zonal statistics are also computed for some of the input layers, specifically the k-factor raster and the slope raster, to determine the average and other metrics of slope and k factor within each contributing area.

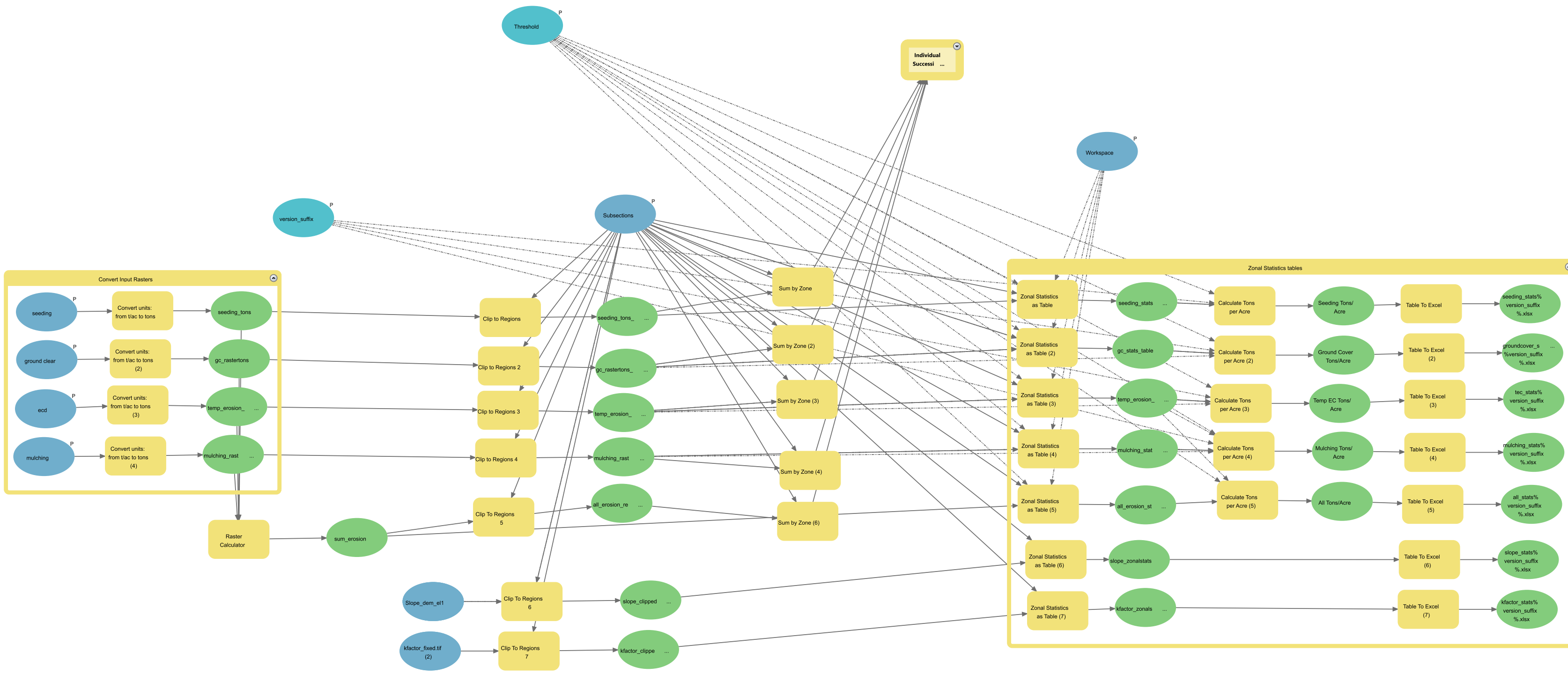


Figure 1: USLE ModelBuilder Postprocessing Routine

### 4.3.1 Contributing area aggregation scheme

the contributing area aggregation scheme describes the amount of sediment in *probable* contributing areas within each ROW. Each contributing area is the region which is within 50 horizontal meters of a flowline which crosses the right of way for the project. Horizontal flow distance was determined from analysis in ArcGIS Pro with the `flow distance` tool. The `flow distance` tool outputs a raster highlighting the flow distance to pre-set flowlines (digitized from LiDAR for the EIS process), which were then thresholded with a raster calculator logic function. The 1s in the boolean logic function were then converted to polygons which became the contributing areas to input into the main function described in Section 4.3. contributing areas derived this way were manually cleaned before being input into the USLE function to remove any contributing areas which were artifacts from looking at the larger raster surface. This included contributing areas which were too small, associated with HDD sites, or which were not directly connected to stream areas.

### 4.3.2 Milepost-based Aggregation Scheme

The milepost-based aggregation scheme describes the amount of sediment discharge by $\frac{1}{10}$–mile region of the proposed construction right of way. Milepost polygons were constructed by producing evenly-spaced points along the centerline of the proposed ROW (spaced at $\frac{1}{10}$ of a mile) and constructing Voroni polygons for the convex hull of the project region. This evenly split the area of the right of way into segments, with some artifacts in areas with less narrow linearity (for example, around valve stations, access roads, and HDD pullbacks). Each milepost marker was then labeled similarly to those in Section 4.3.1 and used as the contributing areas for the main postprocessing function described in Section 4.3. Milepost-based aggregation allows for continuous estimation. These areas were not directly filtered for HDD locations, so they include estimates for some areas which will be drilled under.

# 5 Limitations

Neither modeling scenario will perfectly capture the true sheet flow and rill erosion impact, due to several limitations with the modeling approach. Real-world impact time will vary in between the short and long scenarios depending on how long access needs to be maintained to each area of the proposed ROW and how long it takes crews to work at each individual crossing. For example, HDD sites and adjoining areas will be exposed to longer periods of disturbance than typical upland areas. Modeling also assumes that all areas are impacted by the same amount of rainfall over the same target time period, but this will not be the case during actual construction, where dry or wet spells could increase or decrease the risk of erosion in certain areas. Soil characteristics are much more site-specific than described by SSURGO, so there is some variability in this factor as well. The model was also run including areas where slope exceeded 20% grade, which is the bound of accuracy for USLE. Average rainfall conditions used by the model do not account for the potential of a rarer, more powerful rain event (for example, a five-year or greater return period storm) to impact sediment loss (although this case is not covered by regulation, which only covers up to two-year return interval storms by design). Finally, the modeling tool is limited to factoring in one standard erosion control practice, which limits its ability to fully characterize the impact given a fixed erosion control strategy such as the one presented by Enbridge's EPP and other documentation.

Based on these limitations, outputs of the SLSD/USLE model should be interpreted as relative risk of erosion for each site based on equal starting conditions and uniform land treatment. The model describes how each site would behave if they received the same rainfall and other variable disturbance characteristics, with larger numbers signaling a greater sensitivity to erosion in comparison to other sites. The model is not a prediction of the exact quantity fo sediment that would be eroded by construction activities.

In a regulatory context, USLE is used to determine whether additional erosion control measures are likely to be necessary for an applicant to meet their obligations under state stormwater standards. This legal threshold is 5 tons per acre per year of erosion. For the presently described modeling exercise, this threshold was chosen to signify areas of high erosion risk from construction, sufficient for extra attention from regulators and decision makers when reviewing plans for these areas.

# 6 Results

Figure shows the distribution of each parameter for each scenario. Most crossings fall into the smallest part each distribution, and nearly the entire distribution of the short scenario falls in the first part of the large scenario distribution. Scenario timing and length in general have a large effect on the total erosion potential for scenarios, which is driven by potential exposure to more rainfall and thus more sediment discharge events over the course of the construction project.

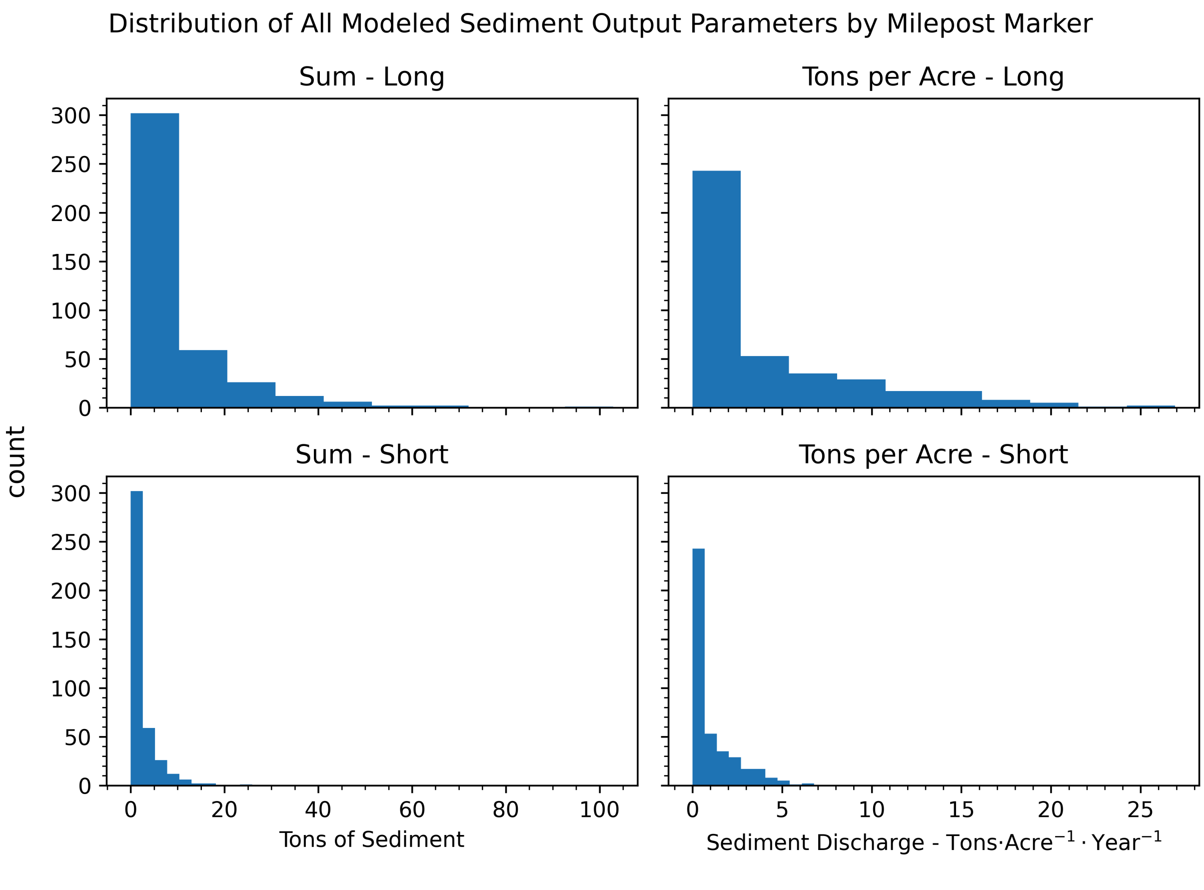


Figure 2: Histogram of outputs by milepost marker from USLE model runs. Each sum denotes the total modeled discharge, while each 'Tons per Acre' denotes the discharge rate on a yearly basis.

Total sediment loss over the modeling period closely follows the distribution of contributing area sizes, with larger contributing areas generally yielding larger amounts of sediment over the course of the simulation. The top-yielding sediment discharging areas for both scenarios are shown in Table 1. Both scenarios have the same top-5 distribution with modified total yields.

The top modeled discharge areas are Unnamed Tributary of (UNT) Silver Creek, UNT Feldcher Creek, UNT Scott Taylor Creek, and UNT of Scott Taylor Creek. UNT Silver Creek in particular has four crossings in the top 10, likely motivated by the steep slopes surrounding this area's stream crossings. UNT Montreal Creek, UNT Krause Creek, UNT Gehrman Creek, and UNT Billy Creek form the top 8 crossings by sediment discharge. Crossings near Billy Creek in particular have large total discharge values due to the large area of disturbance in this location.

Table 1: Outputs of USLE Long and Short Model Runs for Crossing Points

| grid code | Feature ID | USGS Name | Area (acres) | Total Discharge (long, tons) | Total Discharge (short, tons) | Tons per Acre (long) | Tons per acre (short) |
|---|---|---|---|---|---|---|---|
| 165 | sasv002e | UNT of Silver Creek | 0.42 | 8.36 | 2.11 | 19.90 | 5.01 |
| 138 | sirb1002e | UNT of Feldcher Creek | 0.44 | 8.55 | 2.15 | 19.21 | 4.84 |
| 169 | sasv018i | UNT of Scott Taylor Creek | 1.16 | 22.10 | 5.56 | 19.02 | 4.79 |
| 149 | sasa071p_x1, sasa071p_x2 | UNT of Silver Creek | 1.24 | 21.31 | 5.36 | 17.25 | 4.34 |
| 179 | sasv012e | UNT of Scott Taylor Creek | 0.44 | 7.49 | 1.88 | 16.83 | 4.24 |
| 195 | sasd1005e | UNT of Montreal Creek | 0.62 | 10.39 | 2.61 | 16.81 | 4.23 |
| 182 | sasv008i | UNT of Scott Taylor Creek | 0.69 | 11.03 | 2.78 | 15.94 | 4.01 |
| 150 | sasd1015p | UNT of Silver Creek | 1.14 | 17.49 | 4.40 | 15.38 | 3.87 |
| 173 | sasv007i | UNT of Krause Creek | 0.84 | 12.76 | 3.21 | 15.18 | 3.82 |
| 166 | sasa007e_x1 | UNT of Gehrman Creek | 0.62 | 9.16 | 2.30 | 14.82 | 3.73 |
| 172 | sasv017e | UNT of Scott Taylor Creek | 0.32 | 4.74 | 1.19 | 14.75 | 3.71 |
| 32 | sird009p | UNT of Vaughn Creek | 0.86 | 11.95 | 3.01 | 13.82 | 3.48 |
| 127 | sasc1014p_x2 | UNT of Billy Creek | 0.74 | 10.21 | 2.57 | 13.77 | 3.47 |
| 125 | sasc028e, sasc026e | UNT of Billy Creek | 2.82 | 38.45 | 9.68 | 13.65 | 3.44 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 104 | sasc1006p | UNT of Brunsweiler River | 0.74 | 9.94 | 2.50 | 13.41 | 3.38 |
| 133 | sird1006e | UNT of Feldcher Creek | 0.22 | 2.93 | 0.74 | 13.18 | 3.32 |
| 76 | sase1015i | UNT of Marengo River | 1.36 | 17.76 | 4.47 | 13.07 | 3.29 |
| 119 | WDH-103 | Feldcher Creek | 0.94 | 12.27 | 3.09 | 13.07 | 3.29 |
| 10 | sasb007i | Beartrap Creek | 1.11 | 14.50 | 3.65 | 13.04 | 3.28 |
| 126 | sasb1002i, sasb1004e, sasc025i_x, sasc025i | UNT of Billy Creek | 5.34 | 68.93 | 17.35 | 12.91 | 3.25 |
| 147 | sasd1017p | UNT of Silver Creek | 1.19 | 14.99 | 3.77 | 12.64 | 3.18 |
| 184 | sasd1001e, sasd1002e | UNT of Bad River | 0.91 | 11.47 | 2.89 | 12.54 | 3.16 |
| 52 | sasd011p | UNT of Marengo River | 1.19 | 14.02 | 3.53 | 11.82 | 2.98 |
| 174 | sasv016i, sasv013i | UNT of Scott Taylor Creek | 1.11 | 12.44 | 3.13 | 11.18 | 2.81 |
| 58 | sasa020i | UNT of Marengo River | 0.12 | 1.35 | 0.34 | 10.91 | 2.75 |
| 19 | sasd013i | UNT of White River | 1.24 | 13.28 | 3.34 | 10.75 | 2.71 |
| 197 | sasd1006e | UNT of Bad River | 0.57 | 6.08 | 1.53 | 10.70 | 2.69 |
| 163 | sasa006e, sasa005e | UNT of Gehrman Creek | 0.15 | 1.59 | 0.40 | 10.69 | 2.69 |
| 117 | sasc1003p_x1 | UNT of Trout Brook | 0.44 | 4.51 | 1.14 | 10.14 | 2.55 |
| 140 | sirb1001e | UNT of Feldcher Creek | 0.20 | 1.97 | 0.50 | 9.95 | 2.51 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 158 | sasw011, sasw011_x2 | UNT of Gehrman Creek | 1.21 | 11.82 | 2.97 | 9.76 | 2.46 |
| 33 | sasc040e, sasc039i | UNT of Deer Creek | 1.21 | 11.23 | 2.83 | 9.28 | 2.34 |
| 1 | sasa1008e, sase006p | UNT of Bay City Creek, Bay City Creek | 1.24 | 11.25 | 2.83 | 9.10 | 2.29 |
| 183 | sasd1003e | UNT of Bad River | 0.22 | 2.00 | 0.50 | 9.00 | 2.27 |
| 141 | sirb010p | UNT of Feldcher Creek | 0.69 | 6.23 | 1.57 | 9.00 | 2.27 |
| 190 | sasc043i_x1, sasc043i_x2 | UNT of Krause Creek | 0.86 | 7.64 | 1.92 | 8.83 | 2.22 |
| 130 | sird1004i | UNT of Feldcher Creek | 0.25 | 2.14 | 0.54 | 8.65 | 2.18 |
| 71 | sirc005e | UNT of Tyler Forks | 0.07 | 0.64 | 0.16 | 8.57 | 2.16 |
| 131 | sird1005i | UNT of Feldcher Creek | 0.17 | 1.39 | 0.35 | 8.01 | 2.02 |
| 24 | sasc041p | Rock Creek | 1.11 | 8.13 | 2.05 | 7.31 | 1.84 |
| 162 | sasa004p | UNT of Gehrman Creek | 0.86 | 5.82 | 1.46 | 6.72 | 1.69 |
| 189 | sasv010i | UNT of Scott Taylor Creek | 0.30 | 1.98 | 0.50 | 6.66 | 1.68 |
| 38 | sasa067e | UNT of Deer Creek | 0.89 | 5.46 | 1.37 | 6.14 | 1.54 |
| 42 | sasa066i | UNT of Deer Creek | 0.59 | 3.30 | 0.83 | 5.57 | 1.40 |
| 146 | sasw005 | Camp Four Creek | 0.89 | 4.94 | 1.24 | 5.55 | 1.40 |
| 31 | sasc037e | UNT of Deer Creek | 0.67 | 3.70 | 0.93 | 5.54 | 1.40 |
| 90 | sase1023e | UNT of Marengo River | 0.96 | 5.27 | 1.33 | 5.47 | 1.38 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 118 | sasc1003p_x2, sasc1001i | UNT of Trout Brook | 2.05 | 9.81 | 2.47 | 4.78 | 1.20 |
| 89 | sasd1022p, sasd1020e | UNT of Marengo River | 0.99 | 4.09 | 1.03 | 4.14 | 1.04 |
| 113 | sasc1009e_x2, sasa1028i | UNT of Brunsweiler River | 0.96 | 3.92 | 0.99 | 4.07 | 1.02 |
| 159 | sase005p_x2 | UNT of Silver Creek | 0.89 | 3.43 | 0.86 | 3.86 | 0.97 |
| 66 | sase1019i | UNT of Marengo River | 1.24 | 4.20 | 1.06 | 3.40 | 0.85 |
| 9 | WDH-100 | UNT of Little Beartrap Creek | 0.96 | 2.70 | 0.68 | 2.80 | 0.71 |
| 171 | sasv006i, sasv004p | UNT of Silver Creek | 0.84 | 2.29 | 0.58 | 2.72 | 0.68 |
| 59 | WDH-102_x1 t, WDH-102_x2 t | UNT of Marengo River | 4.62 | 6.60 | 1.66 | 1.43 | 0.36 |
| 196 | sasa008p | UNT of Bad River | 0.22 | 0.26 | 0.07 | 1.17 | 0.29 |
| 75 | sirb012p | Tyler Forks | 0.57 | 0.56 | 0.14 | 0.99 | 0.25 |
| 69 | sase1020p | Marengo River | 0.67 | 0.65 | 0.16 | 0.98 | 0.25 |
| 61 | WDH-102_x3 | UNT of Marengo River | 0.54 | 0.52 | 0.13 | 0.96 | 0.24 |
| 60 | sira001i | UNT of Potato River | 0.27 | 0.20 | 0.05 | 0.72 | 0.18 |
| 29 | sasc036e | UNT of Rock Creek | 0.59 | 0.35 | 0.09 | 0.59 | 0.15 |
| 74 | sase1011i | UNT of Marengo River | 0.82 | 0.47 | 0.12 | 0.58 | 0.15 |
| 62 | sase1001e | UNT of Marengo River | 1.36 | 0.72 | 0.18 | 0.53 | 0.13 |
| 98 | sasc1005e, sasc1004e_x1 | UNT of Brunsweiler River | 0.57 | 0.27 | 0.07 | 0.47 | 0.12 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 73 | sase1008e | Ditch | 0.52 | 0.24 | 0.06 | 0.46 | 0.12 |
| 3 | WDH-03 | UNT of Little Beartrap Creek | 0.44 | 0.20 | 0.05 | 0.45 | 0.11 |
| 108 | sasa1027e | UNT of Brunsweiler River | 0.42 | 0.18 | 0.05 | 0.43 | 0.11 |
| 132 | sasb1005i | UNT of Billy Creek | 0.37 | 0.15 | 0.04 | 0.41 | 0.10 |
| 57 | sasc013e, sasa021e | Ditch | 0.59 | 0.24 | 0.06 | 0.40 | 0.10 |
| 44 | sasa068e | UNT of Deer Creek | 0.32 | 0.13 | 0.03 | 0.40 | 0.10 |
| 109 | sasc1007e | UNT of Brunsweiler River | 0.57 | 0.22 | 0.06 | 0.39 | 0.10 |
| 6 | sasa1021e | UNT of Little Beartrap Creek | 0.49 | 0.19 | 0.05 | 0.37 | 0.09 |
| 20 | WDH-107_x1 t | UNT of Vaughn Creek | 1.14 | 0.38 | 0.10 | 0.33 | 0.08 |
| 2 | WDH-02 | UNT of Little Beartrap Creek | 0.54 | 0.18 | 0.04 | 0.33 | 0.08 |
| 81 | WDH-15 | UNT of Marengo River | 0.74 | 0.22 | 0.06 | 0.30 | 0.07 |
| 4 | WDH-04 | UNT of Little Beartrap Creek | 0.79 | 0.21 | 0.05 | 0.27 | 0.07 |
| 56 | sasc012e_x1, sasc012e_x2 | Ditch | 0.05 | 0.01 | 0.00 | 0.27 | 0.07 |
| 21 | sasd013i_x | UNT of White River | 0.37 | 0.10 | 0.02 | 0.26 | 0.07 |
| 46 | sasd015i | UNT of Marengo River | 0.37 | 0.09 | 0.02 | 0.23 | 0.06 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 23 | sasa1020e | UNT of White River | 0.22 | 0.05 | 0.01 | 0.23 | 0.06 |
| 7 | sasa046e, sasa047i | Little Beartrap Creek, UNT of Little Beartrap Creek | 0.94 | 0.19 | 0.05 | 0.20 | 0.05 |
| 28 | sasa016e | UNT of Rock Creek | 0.05 | 0.01 | 0.00 | 0.17 | 0.04 |
| 27 | sird011i | UNT of Vaughn Creek | 0.67 | 0.11 | 0.03 | 0.16 | 0.04 |
| 145 | saws006 | UNT of Camp Four Creek | 0.10 | 0.00 | 0.00 | 0.02 | 0.01 |
| 65 | sase1003e | Ditch | 0.22 | 0.00 | 0.00 | 0.00 | 0.00 |
| 34 | sird006e, sird004e, sird005e | UNT of Vaughn Creek | 1.16 | 0.00 | 0.00 | 0.00 | 0.00 |
| 14 | sirc1001e | UNT of Vaughn Creek | 0.20 | 0.00 | 0.00 | 0.00 | 0.00 |

## References


Renard, K. G., Foster, G. R., Weesies, G. A., & Porter, J. P. (1991). *Comparison of the USLE, RUSLE1.06c, and RUSLE2 for Application to Highly Disturbed Lands*.

SCO. (2019). *LiDAR-Derived Countywide DEM for Ashland County, WI 2019*. State Cartographer's Office. https://geodata.wisc.edu/catalog/8544aa82-bcd5-486a-9e65-a0fb7f8129eb

Soil Survey Staff. (2024). *Soil Survey Geographic Database (SSURGO)* [Computer software]. United States Department of Agriculture. https://www.nrcs.usda.gov/resources/data-and-reports/soil-survey-geographic-database-ssurgo

USDA. (2024). *Revised Universal Soil Loss Equation (RUSLE) - Welcome to RUSLE1 and RUSLE2*. USDA,. https://www.ars.usda.gov/southeast-area/oxford-ms/national-sedimentation-laboratory/watershed-physical-processes-research/docs/revised-universal-soil-loss-equation-rusle-welcome-to-rusle-1-and-rusle-2/

## Source Code

Python

```
import arcpy
import arcpy.sa
import arcpy.management
import arcpy.conversion
import pandas as pd
import math
import numpy as np
import time
import typing
import os

start = time.perf_counter()
#reading in the rainfall energy factors
rfactors = pd.read_csv("rfactors.csv")

#cutting out only the base series
base_series = rfactors.query(
    "date >=42006 & date <= 42370"
)

#adding cols to the base series table
base_series['fractional_date'] = (base_series['date'] - 42005) / 365
base_series['doy'] = base_series['date'] - 42005

#reading in kfactors
kfactors = pd.read_csv("erodibilityfactors.csv")
kfactor_raster = arcpy.Raster("kfactor_raster.tif")
kfactor_raster_lexicon = pd.read_excel(r"kfactor_raster_lexicon.csv")
conversion_dictionary = kfactor_raster_lexicon.to_dict()

#read in lc factor constants
lcfactors = pd.read_csv("lcfactors.csv")

#read in intervention factor constants
interventionfactors = pd.read_csv("interventionfactors.csv")

#reading in the raster
test_slope = arcpy.Raster("slope_el5_decimal.tif")
prototype_slope = arcpy.Raster("prototyping_raster2.tif")

#reading in the texture raster
soiltextureraster = arcpy.Raster(r"soiltextureraster2.tif")

def preprocess_kfactor_raster(kfactor_raster, conversion_lexicon):
    information = kfactor_raster.getRasterInfo()
    information.setPixelType('F64')
    output_raster = arcpy.Raster(information)
```

```
      with arcpy.sa.RasterCellIterator({"rasters":[kfactor_raster]}) as kfactor_cells:
          for r,c in kfactor_cells:
              if kfactor_raster[r,c] ==0:
                  output_raster[r,c] = 0.37
              else:
                  output_raster[r,c] = conversion_lexicon['KfactRF915'][kfactor_raster[r,c] - 1]
          return(output_raster)

  proctexturerast = preprocess_kfactor_raster(kfactor_raster, conversion_dictionary)
  proctexturerast.save(r"..\outputs\kfactor_fixed.tif")

  def meters_to_feet(metric_length):
      return metric_length/0.3048

  def evaluate_ls_factor(slope_raster):
      slope_len = meters_to_feet(slope_raster.getRasterInfo().getCellSize()[1]) # cells should be square, makes this pull easy
      outras = arcpy.Raster(slope_raster.getRasterInfo())

      print("calculating land slope factors...")

      with arcpy.sa.RasterCellIterator({"rasters":[slope_raster, outras]}) as slope_cells:
          for r,c in slope_cells:
              slope = slope_raster[r,c]
              if slope == 0:
                  outras[r,c] = 0
              elif slope < 0.01:
                  outras[r,c] = ((slope_len/72.6)**2) * (65.41 * (slope**2) + 4.56 * slope + 0.065)
              elif slope < 0.045:
                  outras[r,c] = ((slope_len/72.6)**2) * (65.41 * (slope**2) + 4.56 * slope + 0.065)
              elif slope > 0.044 and slope < 0.2:
                  outras[r,c] = ((slope_len/72.6)**0.5) * (65.41 * (slope**2) + 4.56 * slope + 0.065)
              elif slope > 0.2:
                  outras[r,c] = ((slope_len/72.6)**0.5) * (65.41 * (0.2**2) + 4.56 * 0.2 + 0.065)

          return(outras)

  # next: Port over the soil loss a value:

  def eval_soil_loss_a(start_date, end_date, kfactorraster, lc_type, slope_raster, default_k_estimate = 0.37):

    # this is looking up all the values or getting the info I need
    start_per_r= base_series[base_series.doy == start_date].iloc[:,1].values[0]
    end_per_r =base_series[base_series.doy == end_date].iloc[:,1].values[0]
```

```
  period_per_r = end_per_r - start_per_r
  annual_r_factor = 100

  output_raster = arcpy.Raster(slope_raster.getRasterInfo())
  ls_factor = evaluate_ls_factor(slope_raster = slope_raster)
  land_cov_factor = lcfactors.query("lctype == @lc_type").iloc[:,1].values[0]

  print("Evaluating base soil loss...")

  with arcpy.sa.RasterCellIterator({"rasters":[ls_factor, kfactorraster, output_raster]}) as ls_cells:
    for r,c in ls_cells:
      kval = kfactorraster[r,c]
      if np.isnan(kval):
        kval = default_k_estimate
      output_raster[r,c] = period_per_r * annual_r_factor * kval * ls_factor[r,c] * land_cov_factor
  return(output_raster)

 def eval_sdf(slope_raster, soiltextureraster):
   # initializing each raster:
   sand_no = arcpy.Raster(slope_raster.getRasterInfo())
   silt_no = arcpy.Raster(slope_raster.getRasterInfo())
   clay_no = arcpy.Raster(slope_raster.getRasterInfo())
   sdf = arcpy.Raster(slope_raster.getRasterInfo())

   slope_len = slope_raster.getRasterInfo().getCellSize()[1] #should be square, makes this pull easy
   print("Calculating SDF factors...")

   with arcpy.sa.RasterCellIterator({"rasters":[slope_raster, soiltextureraster, sand_no, silt_no, clay_no, sdf]}) as slope_cells:
     for r,c in slope_cells:
       if slope_raster[r,c] == 0: #if there's nodata in the slope raster, set all outputs to 0.
         sand_no[r,c] = 0
         silt_no[r,c] = 0
         clay_no[r,c] = 0
         sdf[r,c] = 0
         continue
       else:
         for i in range(0,3): #this is presently hardcoded, could maybe be more general
           if i == 0:
             if slope_raster[r,c] < 0.045:
               if slope_len < 25:
                 sand_no[r,c] = 1
               else:
                 sand_no[r,c] = 2
             else:
```

```
                            if slope_len < 301:
                                sand_no[r,c] = 3
                            else:
                                sand_no[r,c] = 4
                    if i == 1:
                        if slope_len < 20:
                            if slope_raster[r,c] < 0.045:
                                silt_no[r,c] = 1
                            else:
                                silt_no[r,c] = 2
                        else:
                            if slope_raster[r,c] < 0.011:
                                silt_no[r,c] = 3
                            else:
                                if slope_raster[r,c] < 0.1:
                                    silt_no[r,c] = 4
                                else:
                                    silt_no[r,c] = 5
                    if i == 2:
                        if slope_len < 40:
                            if slope_raster[r,c] < 0.045:
                                clay_no[r,c] = 1
                            else:
                                clay_no[r,c] = 2
                        else:
                            if slope_raster[r,c] < 0.025:
                                clay_no[r,c] = 3
                            else:
                                if slope_raster[r,c] < 0.06:
                                    clay_no[r,c] = 4
                                else:
                                    if slope_raster[r,c] < 0.011:
                                        clay_no[r,c] = 5
                                    else:
                                        clay_no[r,c] = 6

            if soiltextureraster[r,c] == 2: #I would rework this to a switch if I were using a more recent version of python, at least at the top level and maybe at lower levels.
                if sand_no[r,c] == 1:
                    sdf[r,c] = (16*(0.03*slope_raster[r,c]))
                elif sand_no[r,c] == 2:
                    sdf[r,c] = (0.95-(5*slope_len/300) * (0.02 - slope_raster[r,c]))
                elif sand_no[r,c] == 3:
                    sdf[r,c] = (0.79 - 2.5*(slope_raster[r,c] - 0.09) + 0.004 * slope_len * slope_raster[r,c])
                else:
                    sdf[r,c] = (0.955 + 0.005*(slope_raster[r,c]-0.045)*(slope_len-300) - 0.0001*(slope_len-300))
            if soiltextureraster[r,c] == 3:
```

```
                if silt_no[r,c] == 1:
                    sdf[r,c] = (-917.83*(slope_raster[r,c]**2) + 48.312*slope_raster[r,c] + 0.4725)
                elif silt_no[r,c] == 2:
                    sdf[r,c] = (0.1845*(slope_raster[r,c]**-0.589))
                elif silt_no[r,c] == 3:
                    sdf[r,c] = ((0.5317+60.49*slope_raster[r,c])*slope_len**(8.65*slope_raster[r,c]-0.1653))
                elif silt_no[r,c] == 4:
                    sdf[r,c] = (0.66*slope_len**(0.0834)-4.2*(slope_raster[r,c]-0.04))
                elif silt_no[r,c] == 5:
                    sdf[r,c] =(0.3682*(slope_raster[r,c]**0.1649))
            if soiltextureraster[r,c] == 4:
                if clay_no[r,c] == 1:
                    sdf[r,c] =(0.206*math.log(slope_raster[r,c])+1.7385 + 0.00005*slope_len+10*(slope_raster[r,c] - 0.02) - (4*slope_raster[r,c]-0.04))
                elif clay_no[r,c] == 2:
                    sdf[r,c] =(28.087*slope_raster[r,c]**2-8.0411*slope_raster[r,c]+1.3012+5/slope_len-4*(slope_raster[r,c]-0.04))
                elif clay_no[r,c] == 3:
                    sdf[r,c] =(1.1038*slope_len**(-0.095)+28.48*(slope_raster[r,c]-0.002)-0.0006*slope_len)
                elif clay_no[r,c] == 4:
                    sdf[r,c] =((1.5038+3.914*(slope_raster[r,c]-0.025))*slope_len**(-0.045-1.8*slope_raster[r,c]))
                elif clay_no[r,c] == 5:
                    sdf[r,c] =((1.6408-13.342*(slope_raster[r,c]-0.06))*slope_len**(-0.153+1.59*(slope_raster[r,c]-0.06)))
                elif clay_no[r,c] == 6:
                    sdf[r,c] =((0.9737-5.45*(slope_raster[r,c]-0.11))*slope_len**(-0.059+1.1*(slope_raster[r,c]-0.11)))
            if soiltextureraster[r,c] == 0: # treat as a 4, which is the default estimate
                if clay_no[r,c] == 1:
                    sdf[r,c] =(0.206*math.log(slope_raster[r,c])+1.7385 + 0.00005*slope_len+10*(slope_raster[r,c] - 0.02) - (4*slope_raster[r,c]-0.04))
                elif clay_no[r,c] == 2:
                    sdf[r,c] =(28.087*slope_raster[r,c]**2-8.0411*slope_raster[r,c]+1.3012+5/slope_len-4*(slope_raster[r,c]-0.04))
                elif clay_no[r,c] == 3:
                    sdf[r,c] =(1.1038*slope_len**(-0.095)+28.48*(slope_raster[r,c]-0.002)-0.0006*slope_len)
                elif clay_no[r,c] == 4:
                    sdf[r,c] =((1.5038+3.914*(slope_raster[r,c]-0.025))*slope_len**(-0.045-1.8*slope_raster[r,c]))
                elif clay_no[r,c] == 5:
                    sdf[r,c] =((1.6408-13.342*(slope_raster[r,c]-0.06))*slope_len**(-0.153+1.59*(slope_raster[r,c]-0.06)))
                elif clay_no[r,c] == 6:
                    sdf[r,c] =((0.9737-5.45*(slope_raster[r,c]-0.11))*slope_len**(-0.059+1.1*(slope_raster[r,c]-0.11)))

```

```
    return(sdf)

# evaluating the final sediment runoff factor based on the sdf & pev. information:
def eval_final_sediment(sdf_raster, soil_loss_a_raster, interventionfactor, rasterinfo):

    output_raster = arcpy.Raster(rasterinfo)

    print("Evaluating final sediment value...")

    with arcpy.sa.RasterCellIterator({"rasters":[sdf_raster, soil_loss_a_raster, output_raster]}) as slope_cells:
        if interventionfactor == "":
            for r,c in slope_cells:
                output_raster[r,c] = sdf_raster[r,c] * soil_loss_a_raster[r,c]
        else:
            effectiveness = interventionfactors.query("Intervention == @interventionfactor").iloc[:,1].values[0]
            for r,c in slope_cells:
                output_raster[r,c] = (1-effectiveness) * soil_loss_a_raster[r,c] * sdf_raster[r,c]
    return(output_raster)

# a wrapper for the other functions to roll the whole thing into one call easily.
def sediment_assessment(slope_raster, start_date, end_date, soiltextureraster, kfactorraster, lc_type, interventionfactor):

    raster_information = slope_raster.getRasterInfo()

    soil_loss_a = eval_soil_loss_a(start_date, end_date, kfactorraster, lc_type, slope_raster)
    sdf = eval_sdf(slope_raster, soiltextureraster)
    final_answer = eval_final_sediment(
        soil_loss_a_raster= soil_loss_a,
        sdf_raster=sdf,
        interventionfactor= interventionfactor,
        rasterinfo= raster_information)
    return(final_answer)
```