FINAL REPORT – PART B

Programmer's Guide to UnMES Construction & Spatial Application

SERDP Project MR19-1126

MAY 2022

Sarah Rennie
Johns Hopkins University Applied Physics Laboratory

SERDP
DOD ▪ EPA ▪ DOE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
31/05/2022	SERDP Final Report	9/30/2021 - 9/30/2022

4. TITLE AND SUBTITLE

Programmer's Guide to UnMES Construction & Spatial Application Documentation for UnMES Transition Package to NRL-SSC

Final Report - Part B

5a. CONTRACT NUMBER
21-P-0047

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Sarah Rennie

5d. PROJECT NUMBER
MR19-1126

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Road
Laurel, MD 20723

8. PERFORMING ORGANIZATION REPORT NUMBER
FPS-R-22-1131

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Strategic Environmental Research and Development Program (SERDP)
4800 Mark Center Drive, Suite 16F16
Alexandria, VA 22350-3605

10. SPONSOR/MONITOR'S ACRONYM(S)
SERDP

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
MR19-1126

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This document describes the software required to build, train and use the Underwater Munitions Expert System (UnMES), a probabilistic management tool designed to predict underwater migration and burial of abandoned munitions. UnMES is based on a Bayesian Network framework that relates the geological and hydrodynamic site characteristics to the processes governing munitions' interaction with sediments, waves and currents. Simple deterministic models of causal forces acting on the underwater munitions and the sediment responses have been developed to predict scour burial, onset of motion, and the potential for burial or re-exposure due to bedform migration [Rennie, Brandt and Ligo, 2019, hereafter RBL2019]. These models were implemented in Matlab [Mathworks, 2022] at JHU/APL. Other preliminary models for the estimation of liquefaction burial and total distance migrated are presented, despite lack of full validation. These deterministic models are used to build a data set to train the conditional probability tables forming the core of UnMES.

15. SUBJECT TERMS
UnMES Construction, Spatial Application Documentation, UnMES Transition Package, NRL-SSC

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Sarah Rennie
UNCLASS	UNCLASS	UNCLASS	UNCLASS	63	**19b. TELEPHONE NUMBER** *(Include area code)* 443-778-8178

Programmer's Guide to UnMES Construction & Spatial Application
Documentation for UnMES Transition Package to NRL-SSC
MR19-1126 Final Report Part B

S.E. Rennie
Johns Hopkins Applied Physics Laboratory

May 2022

Table of Contents

Acronyms and Symbols

API	–	Application Programmer Interface
BN	–	Bayesian Network
B	–	burial depth of the UXO
CPT	–	Conditional Probability Table
D	–	Diameter of the UXO
JHU/APL	–	Johns Hopkins University Applied Physics Laboratory
MC	–	Monte Carlo
MR	–	Munitions Response
NRL-SSC	–	Naval Research Laboratory - Stennis Space Center
PDT	–	Probability Distribution Table
SERDP	–	Strategic Environmental Research and Development Program
UnMES	–	Underwater Munitions Expert System
UXO	–	Unexploded Ordnance

Programmer's Guide to UnMES Construction & Spatial Application

Introduction

This document describes the software required to build, train and use the Underwater Munitions Expert System (UnMES), a probabilistic management tool designed to predict underwater migration and burial of abandoned munitions. UnMES is based on a Bayesian Network framework that relates the geological and hydrodynamic site characteristics to the processes governing munitions' interaction with sediments, waves and currents. Simple deterministic models of causal forces acting on the underwater munitions and the sediment responses have been developed to predict scour burial, onset of motion, and the potential for burial or re-exposure due to bedform migration [Rennie, Brandt and Ligo, 2019, hereafter RBL2019]. These models were implemented in Matlab [Mathworks, 2022] at JHU/APL. Other preliminary models for the estimation of liquefaction burial and total distance migrated are presented, despite lack of full validation. These deterministic models are used to build a data set to train the conditional probability tables forming the core of UnMES.

The prototype UnMES Bayesian Network (BN) has been initially implemented using the software package Netica™ [Norsys, 2022]. The variables in a BN are represented by nodes, connected by arrows that symbolize dependent relationships. Variable ranges are discretized into a set of states or bins, and each relationship is characterized by a conditional probability table (CPT) associated with the dependent nodes. The Netica software proved useful used to design network structures, visualize relationships between different variables, verify correctness of the MATLAB training outputs, and run sensitivity analyses.

This report is auxiliary to the Final Report of SERDP Project MR19-1126 titled "Development of a Decision Tool: Underwater Munitions Expert System" [Rennie, 2022, hereafter *Development Report*], which describes the underlying concepts and philosophy of UnMES software design. This document provides technical coding details, and specific instructions for the required steps to train, build and use the expert system BN. Multiple appendices provide additional information on the steps involved as guidance for further development and improvement of UnMES.

1) Matlab Code to Build Training Cases

Physics-based models for scour burial and munitions migration on a sandy wave-driven coast have been described in Rennie *et al.,* [2017] and RBL2019. Additional processes, including burial and re-exposure by bedform migration, high-velocity impact penetration, and liquefaction were discussed in Rennie and Brandt [2017, 2019]. Wrapper code to exercise these models in a Monte

Carlo fashion was written in Matlab, in order to output a large set of cases used to train the conditional probability tables of the UnMES Bayesian Network (BN). Matlab was not the optimal choice from the standpoint of execution time, but was well suited to rapid development, testing, and modification.

The code for subroutines to compute scour burial, based on Friedrichs *et al.,* [2016], was documented in Appendix D of RBL2019 for the Prototype Version V1 of UnMES, along with code for ad hoc estimation of munitions migration distance. Initial code for a preliminary liquefaction burial model was presented also in that report. This model was revisited in Rennie and Brandt [2021] with minor improvements. The Monte Carlo (MC) wrapper code, with discussion of the several choices to control the prior distributions for environmental input not previously published, is discussed below.

1.1) Versions of UnMES Bayesian Network for shoaling/surf zone with sandy sediment:

The Prototype Version 1 (V1) of UnMES was documented in RBL2019. Given inputs representing the peak wave forcing for a storm, this UnMES BN predicts burial and migration probabilities from that event at a single location. Several processes are modeled, including impact burial and bedform migration, that inform the probable initial burial state prior to the storm event. Environment inputs for V1 are the peak significant wave height reached during the storm, the dominant wave period at the peak wave, and the concurrent depth-averaged current speed (Figure 1.1a). The node Erosion|Accretion represents farfield seabed changes, e.g. seasonal shoreline adjustment [Rennie and Brandt, 2019], affecting the initial burial.

The nearbed orbital wave velocity and current flow are combined into an intermediate variable represented by the node Um. In UnMES V1, waves and currents are considered over large scales, such that the current direction bins were designed for flows constrained to be largely alongshore. The other required variables for this calculation are the water depth (a fixed value) and the Wave-Current_angle node. The nearbed combined flow is estimated for the local province by linear methods. Process model results for burial and mobility are very sensitive to variation in Um.

An alternate version of UnMES was designed for integration with the Delft3D regional model being implemented at USGS/NRL-SSC for example remediation sites under SERDP Project MR-2733 [Palmsten and Penko, 2020]. Delft3D is a set of numerical hydrodynamic modules developed and supported by the Dutch institute DELTARES [2022]. Delft3D outputs are available on a much higher-resolution grid than the V1 "province" spatial approach proposed in Section 5 of RBL2019. When resolving small-scale spatial patterns as a timeseries throughout the storm, the current flow can take on any direction, even onshore or offshore as when rip currents form. The Delft3D model grid is defined over the entire site, encompassing a range of water depths, and necessarily has to compute the combined nearbed wave-current velocity vectors.

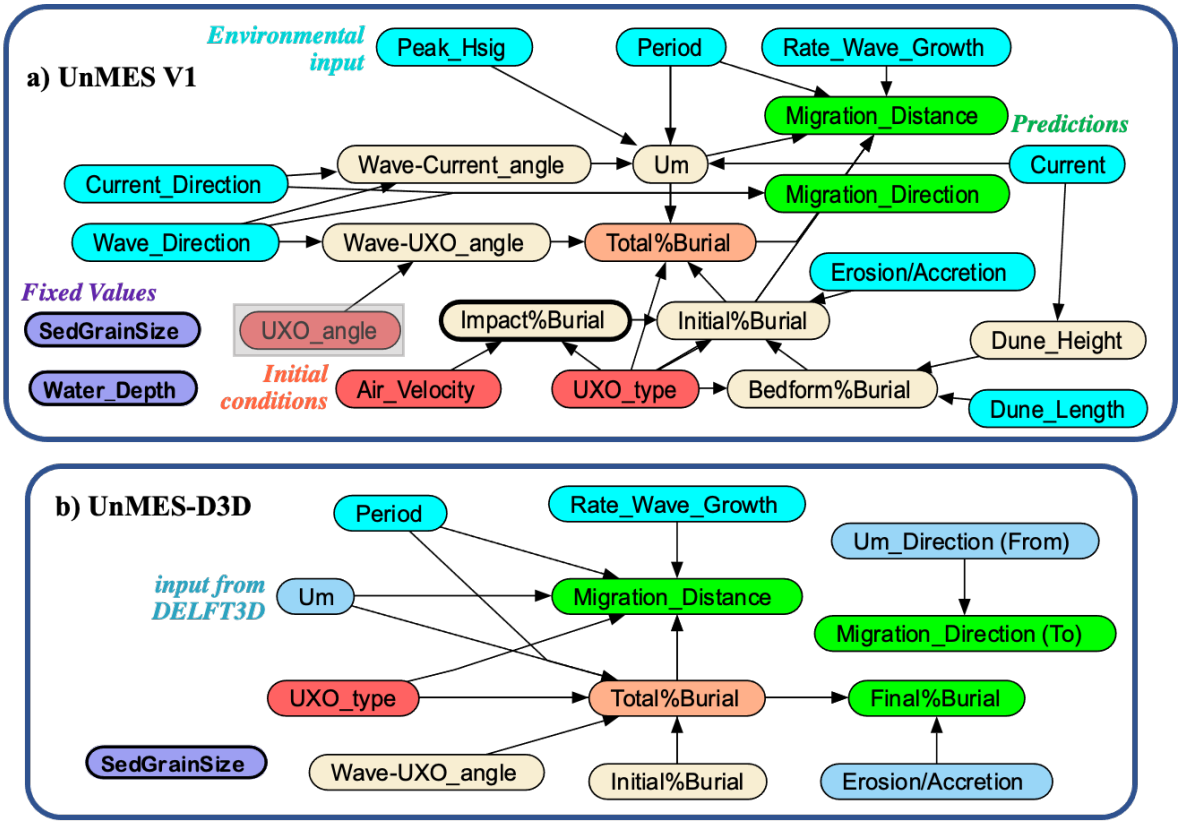

Figure 1.1 a) Bayesian Network for UnMES Prototype V1. b) Bayesian Network UnMES-D3D for use with Delft3D model results. Shaded node in a) is deleted from V2 network.

To support optimal connection with Delft3D, a customized version of the expert system (UnMES-D3D) was built whose use is being investigated by USGS/NRL-SSC [Palmsten *et al.,* 2021]. In UnMES-D3D (Figure 1.1b), Um becomes an input node, so that inputs PeakHsig and water depth are no longer required. A new input node for Um_Direction is introduced, replacing the Wave_Direction and Current_Direction nodes. The Wave-UXO_angle node (see Section 1.2) is retained in this version, although as more experience is gained making use of Delft3D input, that may change. The wrapper code to implement the Delft3D - UnMES connection is discussed in Section 3.

For simplicity, the impact burial and bedform migration burial nodes are removed from this version of UnMES-D3D. Also, because Delft3D calculates the local erosion or accretion at each grid cell when using the coupled morphology module, the function of the Erosion|Accretion node is changed to be a direct additional input, modifying Total%Burial to predict a Final%Burial probability distribution.

Here a new version of UnMES V1 is introduced, documented as Prototype V2. A number of small changes to clarify and improve the V1 BN are made, reflecting experience gained since RBL2019. During UnMES development, the choice of network nodes, their ranges, and bin discretization were frequently made to align with SERDP field and laboratory research experiments in order to demonstrate successful implementation. However, some aspects of the research experiments are incompatible with the design of a realistic, practical expert system. For example, in several field tests, surrogate UXO were deployed by divers at pre-determined angles to the flow, in order to test the effect of varying angle of attack. This information was retained in Prototype V1 UnMES as the input node UXOangle. In an operational setting, the exact angle of abandoned munitions on the seabed is generally unknown, therefore that node is eliminated in V2. Also, the original range of significant wave height (PeakHsig) had a minimum of 1 m, since the research focus was on energetic storms. Operationally, it is more helpful to have the BN nodes' bins cover the full range of possible values, so the PeakHsig node range was expanded. Several other changes made for V2 are noted in the code comments.

In order to implement Bayesian inference in Netica, every node must be discrete with a finite number of states. Some nodes are naturally discrete, e.g., UXO type. However most environmental variables are inherently continuous, so that the total range of the variable must be broken into a number of bins, or states. The bin widths should be small enough to discriminate different response states, but not so narrow that the states have higher resolution than is warranted by the available information. Too many bins can increase the computational requirement prohibitively. Note that a node's bins do not have to be all the same width. It is important to have the baseline Netica Bayesian Network fully delineated with all discretization choices and node connections determined prior to finalizing the customized training code, as several settings within the code must match the Netica values exactly. In particular, the header line of the output text file of training cases must define each column using the Name of the node (not the Title, see format description in Section 1.2).

Munitions studies in non-cohesive sediments reveal that there is counteraction between scour burial and onset of motion. Even a small amount of burial suppresses the munitions' ability to mobilize. Analysis of field data by Traykovski and Austin [2017] suggested that it is important to correctly predict the time-dependent evolution of scour burial, and that the timescale for burial must be substantially longer than previously reported from laboratory work, e.g. Demir and Garcia [2007], discussed in Rennie *et al*. [2017]. Traykovski suggests a timescale O(10) longer; the best value to use is still under investigation.

Given the time dependence of scour, and the temporal competition between burial and mobility, the rate at which the waves and currents increase, i.e. the growth speed of the storm, is an important factor. The node Rate_Wave_Growth (Name: dHsdt) was added to the Prototype UnMES BN,

representing the maximum time derivative of significant wave height, (dH_{sig}/dt). With this approach, when creating case files to train the expert system BN, it is necessary to model not just the peak of the storm, but the storm sequence as a time series. Part of the MC simulation must include repeated random sampling of synthetic storm patterns.

The names of Matlab routines for the deterministic physics-based process models are listed in Table 1. Explanation of the equations and variables are found in RBL2019. Additional routines based on empirical relationships derived from oceanographic studies, as well as standard hydrodynamic equations, are also included in Table 1. Creation of temporal storm patterns is implemented in the function `storm_duration_shape.m`, based on classification of event duration by Dolan and Davis [1992], combined with schematized temporal shapes proposed in Poelhekke *et al*. [2016].

1.2) Monte Carlo Wrapper Code to build training dataset: deterministic process models

The wrapper code written to execute a MC exploration of scour, liquefaction burial, and migration distance is initiated with `script_maketrainingcasesProtoV2UnMES.m`., which produces output to train the BN nodes Combined_Wave-current_Amp (Um), Total%Burial (TB), Migration_Distance (Mdist), as indicated in Table 1. Separate scripts are run to produce training cases for the nodes Impact%Burial (ImpB) and Bedform%Burial (BB), discussed in Sections 1.4 and 1.5. A single training set for V2 will cover all UXO types of interest for one location, for a given water depth and fixed sand grain size (see discussion of Mode 1 operation in RBL2019, Section 5).

Table 1. Matlab Code file names

Physics-based or Empirical Functions	
`lag_scour_liquefy_2019.m`	`scour_demirgarcia_timescale.m`
`ZLiquefaction_ProtoV2.m`	`bound_simple1dhydroIDA.m`
`storm_duration_shape.m`	`SandiaPonceletPenetration.m`
`bottom_orbital_vel.m`	`shield_sed.m`
`scour_demirgarcia_timescale.m`	`duneheight_vanRijn.m`
`bottom_orbital_vel.m`	`fixedSawtoothDunePercentageBurial.m`
Monte Carlo Wrapper Code	**Output Trains BN Node Title (Name)**
`script_maketrainingcasesProtoV2UnMES.m`	Combined Wave-currentAmp (Um)
`mc_scour_liquefy_migrate_ProtoV2.m`	Total%Burial (TB)
`Init_UXO_Type.m`	Migration_Distance (Mdist)
`scriptMCimpactPenetration.m`	Impact%Burial (ImpB)
`scriptMC_BB_BedformHeight.m`	Dune_Height (DuneHt)
Code to Directly Build PDT	
`script_BedformBedformPDT.m`	build CPT for Bedform%Burial (BB)

The script sets the ranges from which uniform draws are pulled for values of the hydrodynamic input variables: current speed, wave height, wave period and dH_{sig}/dt. There is no joint distribution imposed between wave height and period in this implementation. The minimum and maximum of the ranges must match the discretization extrema of the hydrodynamic input nodes in the Prototype BN. Note that the probability distribution table (PDT) for an input node in UnMES does not represent a meaningful prior distribution, as is often the case in Bayesian applications, but simply shows the MC coverage of the domain of interest. The effect of the angle variables ("beta" for the wave-current angle, and "alpha" for wave-UXO angle) are not sensitive to small variations, and are therefore covered in the MC simulation by single representative values from each the 3 states of the angle nodes ("parallel" , "angled", and "perpendicular").

The main routine called is `mc_scour_liquefy_migrate_ProtoV2.m`, which is called once for each UXO type. The text case files for training are output separately for each UXO to keep the file size manageable. The case file format required by Netica is very simple: ascii text columns with one column for each node that is referenced. Each case is a single line in the file. The first line of the file must be a Header line specifying the Node Names (not Titles) as used in the Netica BN to be trained, separated by space and/or tabs, in an order matching the columns. The column values can be numeric or text, depending on the corresponding node's states. Table 2 displays a set of lines from an example training file as created by the Matlab wrapper code for scour burial. The Header line is in blue, giving the Node Names for each column. Above, in purple, are the matching Node Titles, as shown in Figure 1.1a. Only the BN nodes involved in the training are included. For example, the Impact%Burial node (Name: ImpB) is not part of the process model to predict scour burial, liquefaction, or migration, because its information has been incorporated into the Initial%Burial (Name:IB) value, therefore a column for ImpB is not included in this case file.

Table 2. Example Case File

Node Title	Peak_ Hsig	Period	Current	Wave-UXO_ angle	Wave-Current_ angle	Um	Initial% Burial	Rate_Wave _Growth	Total% Burial	Migration _Distance	Water_ Depth	Sed Grain Size	UXO_ type
Header line	Hsig	T	Uc	alpha	beta	Um	IB	dHsdt	TB	Mdist	hDepth	dsed	UXO
case 1	3.3	10.2	0	10	5	2	0.27	0.51	0.63	9.1	3	fine	howitzer
case 2	6.5	7.4	0.04	10	5	1.9	0.85	1	2.19	0	3	fine	howitzer
case 3	8	8.1	0.51	10	5	2.26	0.87	0.59	2.63	0	3	fine	howitzer
case 4	3.3	11	1	10	5	2.7	0.71	1	1.04	0	3	fine	howitzer
case 5	1	4.1	0.98	10	5	1.27	0.09	0.35	0.43	0.2	3	fine	howitzer
case 6	4.8	10.2	0.88	10	5	2.69	0.07	0.96	1.19	82.7	3	fine	howitzer

Note that Netica allows that specification of a case with missing values, as a Bayesian Network is designed to function with uncertain or unavailable data. In a Netica case file, the missing value is designated with an asterisk. For the MC exploration of deterministic models, there are no missing values in the training cases. However, the test cases discussed in Section 5 of the *Development Report* [Rennie 2022], taken from field experiments and used for skill assessment, had several unrecorded quantities that were marked *.

For improved execution speed at JHU/APL, the different UXO types are modeled in a parallel (a `parfor` loop), making use of the Matlab Parallel Computing Toolbox. However, simple sequential modeling is possible also. Multiple naming and organizational choices are hardcoded in this routine which must match up correctly with the Netica BN that its output is intended to train. The number of Monte Carlo samples to draw is the hard-coded value `nmcsamp`; how many draws are sufficient depends on the number of bins in the parent and child nodes involved. The angle variables are iterated in nested `for` loops (3 states each) outside of the `nmcsamp` draws, so that the total number of training cases generated is `3*3*nmcsamp`. For Prototype V2, `nmcsamp` of O(10^5) will demonstrate the general physics; sampling effects will be less with larger numbers. Within Matlab execution, writing text to the output file is most time-consuming portion of the run; therefore the cases are stored in a large array and output all at once at the end. For very large values of `nmcsamp`, this could cause problems with allocated memory. A schematic illustrating the MC procedure to build the training dataset for UnMES is shown in Figure 1.2.

The routine `mc_scour_liquefy_migrate_ProtoV2.m` generates `nmcsamp` storm sequences. The peak wave height H_{sig} for a given sequence is drawn from a uniform distribution of allowable heights given the local water depth d, noting the wave breaking will decrease the upper limit of H_{sig} where d is shallow. Another random draw provides a value for the wave growth rate, dH_{sig}/dt. The duration of the storm sequence is computed by `storm_duration_shape.m`, adjusted to use the wave growth rate, resolved with a time step of 1/2 hour, and imposing a minimum duration for the events with small H_{sig}. Note that for implementations of UnMES with shallow water depth, where the maximum wave height is limited by breaking, the same are used for the `PeakHsig` nodes.

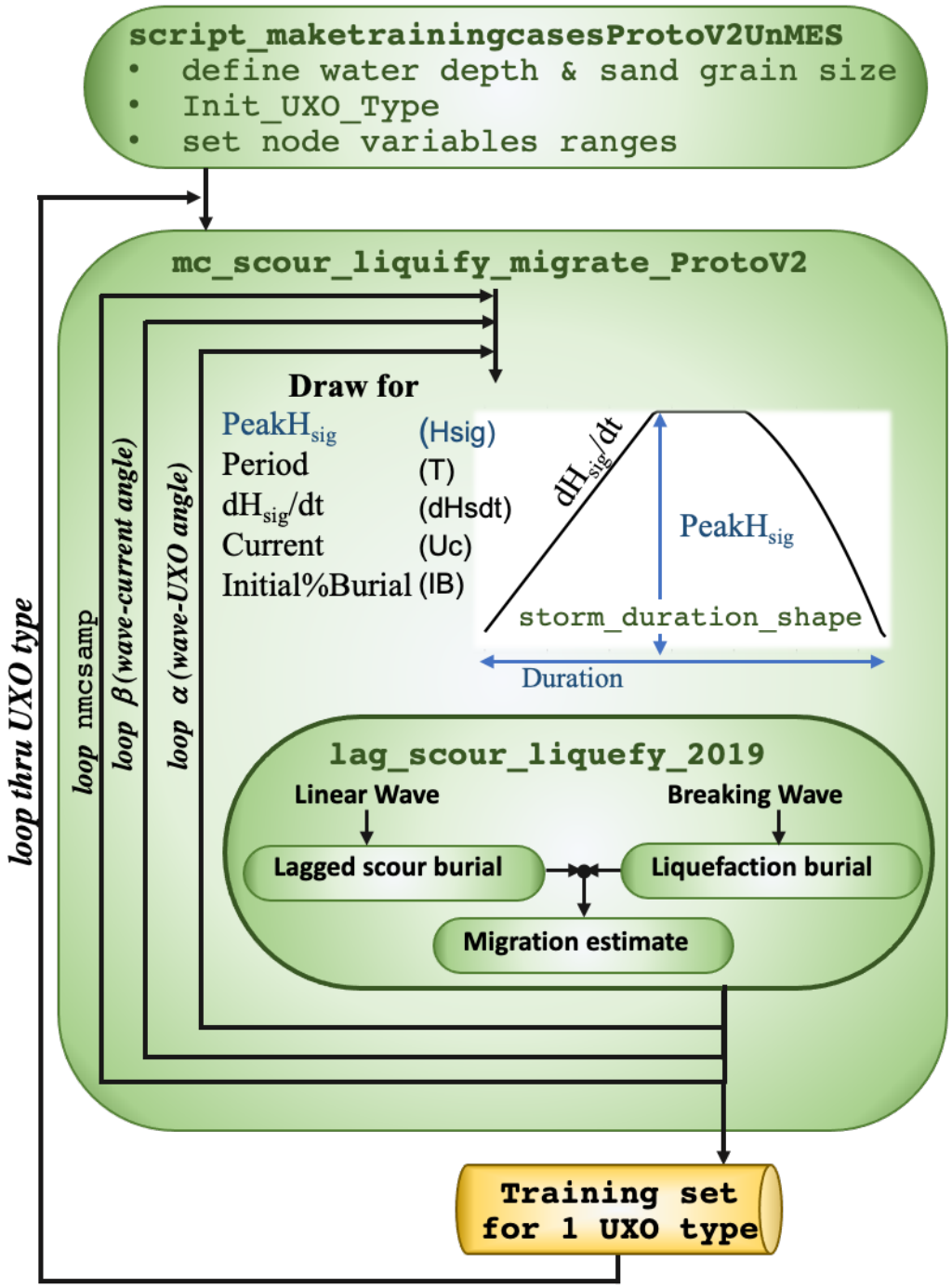

Figure 1.2 Flow chart for Monte Carlo procedure to generate cases to train Prototype UnMES.

1.3) Process models lacking full validation:

During each storm event, scour will generally initiate UXO burial, computed in the function `lag_scour_liquefy_2019.m`. In high energy conditions, the code switches to the liquefaction routine, `ZLiquefaction_ProtoV2.m`, when the Shields parameter θ becomes large (θ > 1)

indicating that sheet flow conditions prevail, or if H_{sig} exceeds the wave breaking criteria. When the high-energy threshold is passed begin, the implemented code passes the scour burial computed up to that point. The liquefaction code should be considered preliminary. As discussed in Rennie and Brandt [2021], the estimation of liquefaction depth is strongly sensitive on sediment properties such as gas content and shear modulus which are poorly known. In the V2 implementation, these properties are not input. Instead, the code treats saturation ratio S_r as a random variable (gaussian with $\mu = 0.99$ and $\sigma = 0.0025$) as in Rennie and Brandt [2021] to illustrate this uncertainty. The shear modulus G is set to 1.125E+7 N/m^2, suggested as the baseline condition in Klammler *et al*. [2020]. This liquefaction routine should be considered a placeholder only, and further improvement to accurately compute high-energy burial in non-cohesive sediments is needed.

Once the timeseries of burial within the storm sequence is computed, the potential for onset of motion is evaluated based on the critical threshold, U_{crit}, described in Rennie *et al.* [2017]. If the mobilizing forces are above threshold, the routine `migrate_estimate_ProtoV2.m` is called. Reasonable validation for onset of motion has been obtained, but our ability to estimate total migration distance continues to be unsatisfactory. The most extensive migration field set was obtained in the surf zone at Long Point, Martha's Vineyard during 2014 [Traykovski and Austin, 2017], where migration distances over 100 m were observed for the less dense UXO. The Prototype process model for migration distance in UnMES assumes than the speed of UXO migration, after onset of motion, varies as $(U - U_{crit})^3$, and relies on scaling coefficients and event duration which were empirically tuned to match the 2014 Long Point migration data.

However, a second field test was conducted at the same surf zone location in 2018 deploying similar surrogate UXO to the 2014 test, with remarkably different migration behavior observed. In spite of stronger wave forcing, very little migration occurred. Analysis by Traykovski and Jaffre [2020] showed a clearly different set of mobility processes was dominant, such as will require a new, more sophisticated approach to model. A new SERDP project, MR21-1341 [Traykovski and Palmsten, 2020], is underway to investigate whether numerical shoaling wave models can successfully predict migration observations under varying conditions. In the meantime, the Prototype V1 migration estimation is still used in UnMES V2, but should be treated as a placeholder.

Similarly, there is little information on which to base a prediction of the direction towards which the UXO will migrate. The 2014 Long Point data showed generally onshore motion with an expected pattern: more alongshore movement opposite from the incoming wave direction. Based on these observations, and statistics from a few other data sets of UXO migration [e.g., Jenkins *et al.*, 2013], an ad hoc table was devised relating wave direction, with some influence from the currents, to the probability of UXO migration direction. Again, this should be considered a placeholder until better understanding of UXO response to hydrodynamic forcing is available.

1.4) Training Cases for Impact Burial for UnMES V2 in sand

Studies previous to SERDP generally found that munitions dumped from ships onto hard sand do not experience any significant burial upon impact [NATO, 2007]. However, at sites such as former artillery ranges in shallow water, there is the potential for munitions to impact the seabed at very high velocities. Ballistic penetration models for burial into sand were explored for the UXO in UnMES [Rennie and Brandt, 2017], and a formulation was developed at Sandia National Laboratories [Young, 1997] was implemented for inclusion in Prototype V1 UnMES. There are on-going SERDP projects focusing on the problem of high-speed penetration in saturated sand [Bless *et al.,* 2020] which will provide much improved models in the near future. In the meantime, the Impact%Burial node in UnMES V2 is trained using the Sandia model.

The MC wrapper code `scriptMCimpactPenetration.m` computes the training cases for impact burial, calling the subroutine `SandiaPonceletPenetration.m`. The velocity at the air-water interface is specified in an input node titled Air_Velocity, (node name: V00) whose distribution would be based on knowledge of historical activities at the remediation site. The change in velocity through the water trajectory to the seabed is computed using a simple hydrodynamic model, `bound_simple1dhydroIDA.m,` that assumes exponential approach to the UXO's terminal velocity [Teichmann et al., 2017]. Terminal velocity varies with the angle of attack, which is treated in a probabilistic manner. The training file output contains cases for all UXO in the expert system.

At the moment, Water Depth is treated as a scalar (constant) node in Prototype V1 and V2 UnMES, but could be implemented as a PDT in a future version. For both impact and bedform burial, the water depth value hDepth is varied stochastically within the defined bins, which are bounded as Water_Depth (hDepth) as `[2,2.5,3,4,5,6,8]` meters. The variation in burial predictions within a depth bin is insignificant for the deeper bins, but can be important for hDepth < 3m.

1.5) Direct Computation of Bedform Burial CPT

An alternate approach to building the Conditional Probability Table was used for the Bedform%Burial (BB) node. Several observations of UXO burial under a migrating dune were reported in a tidal channel by Traykovski and Austin [2017]. The probability of UXO burial or exposure due to current-driven bedform migration was considered in Rennie and Brandt [2019], using a geometric argument assuming the traveling sandwaves have a typical sawtooth profile. MC code `scriptMC_BB_BedformHeight.m` creates training cases for the node Dune_Height (DuneHt), based on an engineering model from Soulsby [1997]; see the subroutine `duneheight_vanRijn.m`. The length of the dunes is treated as an input variable, the node Dune_Length (DuneL).

The formation of the CPT relating DuneHt , DuneL, and UXO_type to percentage bedform burial , illustrates another method of including probabilistic dependencies in a Netica BN. The code script_BedformBurialPDT.m, rather than training case output, writes the CPT directly, counting from a high-resolution simulation (fixedSawtoothDunePercentageBurial.m). Netica provides the ability to hand-enter node Table values, using the GUI. However, for a complex or large table, the table text can be inserted into the ascii format BN file. Details are provided in Appendix D.

The burial probabilities from this geometric analysis of migrating dunes are representative only over large space and time scales, a mismatch to the temporal scale of the rest of the UnMES BN. In particular, the current driving the migrating dunes is considered to being acting over a long time period, not just one storm event. In addition, it is assumed that the difference between the observed and modeled dune length results in gaps between the sand waves, a situation that is not universal, and which has a strong influence on the estimated probability of burial. There is much room for improvement in the expert system's knowledge base of bedform burial and re-exposure. However, at present there is no funded SERDP project investigating this topic.

2) Training UnMES

To achieve an estimate representative of the true conditional probability, many training cases are required. As discussed in Section 3.2 of the *Development Report*, the number of training cases needed to achieve good confidence can exceed several million. To make the computation time less onerous, and to keep the training case file sizes practical, the usual procedure has been to train for each UXOtype separately. Previous MC construction of UnMES used uniform draws across the full domain of the parent node. However, because many nodes are discretized into irregular bins, this results in an uneven distribution of cases in the CPT entries. A segmented specification of the MC draws, customized to match the bin choices, could be programmed into order to more efficiently distribute the computed cases.

2.1) Training UnMES using Case Files

The training cases created by script_maketrainingcasesProtoV2UnMES.m are written to an ascii text file in the columnar format required by Netica, where the header line supplies the node name for each column. Netica's "Learning from Cases" enacts parameter learning by counting to modify the CPT for each child node, given the link structure and the accompanying input data for each case.

These training case files can be pulled into Netica by hand (described here), or applied in an automated manner using code written for the Netica-J Application Programmer Interface (API). Installation of the Netica-J API is detailed in Appendix A.3 and the Matlab wrapper code using the API to train an UnMES BN with cases is shown in Appendix B. Experience has shown that

API behavior under Python is more stable than with Matlab. Therefore, these routines may best be rewritten using Python with use of the Netica-C API for future management.

When working directly through the Netica application GUI, usually the programmer starts with an untrained base network. To remove any previous training, one can select all the nodes and choose `Remove` from Netica's `Table` menu. A training case file is applied to the base BN using Netica's `Cases>Learn>Incorp Case File` menu choices. Netica requests a value for the degree (weighting or multiplicity) to apply to each case; generally the value 1 is used. When working by hand, each UXOtype file has to be incorporated separately; their portions of the CPT will be combined by the learning algorithm. After the first file, a pop-up will ask "Remove existing node CPT and experience tables first?"; be sure to specify `NO`, so that the table entries for the previous UXOtype are retained.

2.2) Training UnMES using Equations

A few of the child nodes in Prototype V2 of UnMES act to combine together their parent's nodes in a simple manner that can be stated in a straightforward equation. Netica allows for the creation of the relation contingency table, or CPT, directly from an equation. Netica equations can be formed following most of the standard mathematical equations, with the usual mathematical operators and simple functions. For example, the node Initial%Burial (IB) in Prototype V2 designates the degree of burial prior to the storm event, and is a combination of burial due to impact penetration into the sediment (ImpB), along with burial under bedforms (BB), and possible burial (or exposure) due to accretion (or erosion) (EA). Its equation is written as

IB (EA, ImpB, UXO, BB) = EA/(UXO/100) + max(ImpB, BB)

Note that Erosion|Accretion is dimensional (meters), while the other burial variables are fractional, therefore EA must be divided by the UXO diameter, which is the UXO_type node value (UXO is in cm). The Erosion|Accretion node is treated differently in the UnMES-D3D version (Figure 1.1b), where the input value from the Delft3D is treated as the morphological change driven by the storm being modeled. In this case, EA is added to the Final%Burial (FB) node.

Prior to compiling the Netica BN, all node with equations must have their tables built. To build tables for particular nodes, select the nodes, and then choose Table → Equation to Table from the Netica menu. A pop-up will ask for the number of samples per cell; the larger this number, the more accurate the results will be, but the longer the computation takes. Netica then asks whether to include sampling uncertainty; for simple well-behaved equations, this is not necessary.

The Wave-Current_angle (beta) node combines the wave (wDir) and current (cDir) directions in the Prototype UnMES to inform both Um and the Migration_Direction node, with the equation

beta (wDir, cDir) = acos(cos(pi/180*(wDir-cDir)))*180/pi

For some situations, even when the relationships between nodes are fairly simple, it is easier to generate a set of training cases, than to use the Netica equation option. For example, Migration_Direction in UnMES-D3D is assumed to be simply the opposite direction from the forcing variable Um_Direction (UmDir). But, with Delft3D directions covering the full compass, handling the wrapping around 0 (shorenormal) was more easily accomplished using a Matlab script to compute cases that map the angle bins.

3) Connecting Delft3D output with UnMES-D3D

An example of a spatially-varying implementation of UnMES has been programmed using the gridded output from the Delft3D model as input values for the environmental nodes, as discussed in Section 4 of the *Development Report*. The regional numerical hydrodynamic model Delft3D has been implemented by SERDP Project MR2733 [Palmsten and Penko, 2020] at several coastal locations representative of recent SERDP UXO studies. Considerable effort is required calibrate the model parameters to achieve good match between the predicted and observed waves and currents. Morphological changes needed to populate the Erosion/Accretion input node are notably challenging to predict with accuracy. Successful operational application of Delft3D-UnMES modeling will require substantial investment in data gathering and model tuning. In particular, the local shoaling bathymetry approaching the shoreline needs to be well-resolved.

Choice and management of successively finer (nested) grids is an important factor in Delft3D stability and calibration for a specific site. The outer (lower resolution grid) simulates the offshore waves, followed in some implementations by a middle (moderate resolution) region. The innermost highest resolution grid for waves is forced by the outer grid results at its boundary. Then the "Flow" domain on another high resolution inner grid is coupled with the inner wave simulation to model the currents and morphology. Afterwards, a post-processing step transfers the Delft3D simulation output from several different inner grids into Matlab structures, which are then reprocessed into UnMES compatible cases using the script `make_network_input_D3D.m`.

3.1) Translation of Delft3D to UnMES-D3D Cases

The present `make_network_input_D3D` script, listed in Appendix E, was developed originally by the USGS/NRL-SSC team, and modified at JHU/APL. Multiple hard coded values are customized to the specific Delft3D output, in this case a simulation of the hydrographic conditions at the US Army Corps of Engineers (USACE) Field Research Facility (FRF) at Duck, North Carolina (see Section 4 of the *Development Report*). In particular, the choice is made to use the FRF Northing, Easting coordinate system as the standard grid to which the other grids are interpolated. The `make_network_input_D3D` script will continue be a work in progress as the Delft3D modeling team improves and reorganizes the simulation output. Continued interaction

with the Delft3D team should streamline the transfer steps and bookkeeping required to coordinate the multiplicity of grids. The Delft3D results required for input to UnMES are found in the Matlab structures listed in Table 3, based on the format as currently supplied from the model output routines. All time steps are output; here, the `make_network_input_D3D.m` script selects out only the time step of interest (the peak of the storm, itpeak).

Table 3. Variables from Delft3D for Transfer to UnMES-D3D

Structure Name	Fields of Interest	Values	Units	Used to compute UnMES: Node Title	Used to compute UnMES: Node Name	Comments
depth		bathymetry	m	Water_Depth	hDepth	2-D matrix
wave	.Lon	Longitude of wave grid location	deg			on coarser grid than the velocities
wave	.Lat	Latitude of wave grid cell location	deg			
wave	.Hs	significant wave height	m	Peak_Hsig	Hsig	
wave	.Tp	wave period	s	Period	T	
wave	.Ub	orbital velocity near bottom	m/s	Um	Um	
vel	.data.X	Longitude and Latitude of depth-averaged current grid cell location	deg			X is Longitude
vel	.data.Y		deg			Y is Latitude
vel	.data.Val	depth-average current magnitude	m/s	Current	Uc	
vel_vec	.data.X	Longitude and Latitude of current components grid cell location	deg			same size grid as vel.data, slightly offset (corners for vel calculation)
vel_vec	.data.Y		deg			
vel_vec	p	depth-aver current E-W comp	m/s	Um_Direction	UmDir	
vel_vec	p	depth-aver current N-S comp	m/s	Um_Direction	UmDir	
eros	.data.X	Longitude and Latitude of erosion grids from MORPH module	deg			on same grid as current
eros	.data.Y		deg			
eros	.data.Val	cum. erosion/sedimentation	m	Erosion/Accretion	EA	

The current is computed in Delft3D on a higher resolution grid than the waves, and all grids are in latitude, longitude coordinates. It was decided for UnMES that all values should be interpolated onto the current grid cell locations, and transformed into the FRF Northing, Easting coordinate system so that distances and angular directions are easier to calculate. Much of the code in the `make_network_input_D3D` script is devoted to the bookkeeping involved in rearranging and interpolating from the 3-dimensional matrices (Lat, Lon, time) onto 2-dimensional cross shore and alongshore grids (XCross and YAlong). A new input node for Um_Direction is introduced in UnMES-D3D, replacing the Wave_Direction and Current_Direction nodes. Particular care must be taken to translate the wave and current compass directions into the right-handed shorenormal angle convention used by UnMES. In addition, values exceeding the bin ranges defined for the UnMES input nodes need to be controlled. Comments in the code listing in Appendix E provide more details.

The Delft3D model supplies a depth-averaged current vector. The `make_network_input_D3D` script estimates the near-bed current components by assuming a logarithmic velocity profile (law of the wall) . The bed roughness length requires a value for the representative sand grain size (dsed)

and assumes hydrodynamically rough flow [Soulsby, 1997]. For input to UnMES-D3D, the script computes Um and Um_Direction by combining the near-bed wave and current vector components, rotated into UnMES shoreline oriented coordinate system. As increased experience is obtained with additional site implementations, modifications to this script will be needed to adjust several of the hard-coded choices described here, such as the sand grain size (dsed), and the time step of interest (itpeak). Below are detailed the steps required to prepare Delft3D gridded results for input to UnMES as a Netica-format "case" text file, here called `DuckExampleD3D.txt`.

Steps 3.1 from Matlab script `make_network_input_D3D.m`

1) put all Delft3D output onto common grid
2) limit Delft3D output values to UnMES-D3D node bin ranges
3) transform directions to conform to UnMES shore-normal coordinate system
4) adjust depth-averaged current to near-bottom flow
5) combine current and wave near-bottom velocity vectors
6) output each grid location as a "case" in Netica-format text file

3.2) Python wrapper for UnMES spatial application

The next step is performed by the Python script `UnMES_D3Dconnect.py`, which takes each case from the text file created in the previous section, and sends the input values to the corresponding UnMES-D3D nodes. The resulting probability distribution tables (beliefs) from the output nodes (e.g. Final%Burial or Migration_Distance) are then retrieved using the function `queryNetCase`. These probability distributions can be further processed for visualization, such as a on Google Earth map. Methods to display multi-dimensional probabilistic results on maps are discussed in Section 4 of the *Development Report* and in Rennie and Brandt [2020].

When setting up the Python environment for `UnMES_D3Dconnect.py`, installation of Python 3.7.3 or higher is required, as well as the capability to compile C++ code within Python. Several extensions such as C/C++ and YAML are needed. Installation of the *pynetica* package, developed at NRL-SSC by S. Bateman, is needed to provide access to the Netica-C API functionality. Note this should not be confused with a different Python package also called *pynetica* available from DELTARES. In addition, the library from Netica, called libnetica.a, must be available in the Python environment site-packages area. Other Python packages from the geoscience community such as simplekml and gearth have been useful for creating georeferenced visualizations. However, future development may prefer more recent Google Earth interfaces.

When run, the first task of `UnMES_D3Dconnect.py` is to create the Netica environment by establishing a link with the current version of UnMES-D3D BN, named `Prototype2022_UnMES-D3D_fine.neta`. The ending string in the file name designates that this BN was trained for the representative sediment size in the range of "fine" sand grains (0.10 <

dsed < 0.26 mm). The BN is compiled and all PDTs set to their "no knowledge" values. Then the `DuckExampleD3D.txt` file is read in, and each case, or the values at each grid location, are passed to the BN using the function `queryNetCase`. After that, various map plots are made in KML format for presentation by Google Earth. In addition, the probability distribution at a particular location of interest is displayed as a histogram bar plot. In this version, many choices, such as case file name and information about the size of the spatial grid, are hard coded. Future versions will need to provide a mechanism to pass that information from the `make_network_input_D3D` stage. A useful upgrade would be to merge the `make_network_input_D3D` functionality into the Python scripts. Appendix E presents the `UnMES_D3Dconnect.py` code listing with extensive comments for further guidance.

Literature Cited

Bless, S, M Iskander, M Omidvar, Robust Prediction of Penetration Depth of Burial in Soils Using a Field Calibrated Phenomenological Model and Probabilistic Simulations, SERDP Project MR19-1277, SERDP/ESTCP Symposium 2020, November 30, 2020, Washington, DC.

Deltares, 2022. https://oss.deltares.nl/

Demir, ST, MH Garcia, 2007. Experimental Studies on Burial of Finite Length Cylinders under Oscillatory Flow, J. Waterway, Port, Coast, and Ocean Eng., ASCE, 133(2),117-124

Dolan, R, Davis, RE, 1992. An intensity scale for Atlantic coast northeast storms. Journal of Coastal Research, pp.840-853.

Friedrichs, CT, Rennie, SE, Brandt, A, 2016, Self-burial of objects on sandy beds by scour: A synthesis of observations, *Scour and Erosion*. CRC Press, 179–189.

Jenkins, S, G D'Spain, J Wasyl, 2013. "Hydrodynamic Mobility Analysis of UXO Transport, Andrew Bay Adak Island, Alaska," Scripps Marine Physical Laboratory Report to Battelle Memorial Institute, 99 pp.

Klammler H, Scheremet A, Calantoni J, 2020. "Seafloor Burial of Surrogate Unexploded Ordinance by Wave-Induced Sediment Instability," *IEEE Journal of Oceanic Engineering, vol. 45, no. 3, pp. 927-936, July 2020.*

Mathworks Software, 2021. https://www.mathworks.com/products/matlab.html

NATO Naval Armaments Group, Maritime Capability Group 3, Mine Countermeasures and Harbour Protection, 2007. Final Report by Specialist Team for Sea Mine Burial Expert System (SMBES), NATO Document AC/141(MCG/3)D(2007)0001, NATO Unclassified.

Norsys Software, 2022. **https://www.norsys.com/netica.html.**

Palmsten, M, A Penko, 2020. "Probabilistic Environmental Modeling System for Munitions Mobility," SERDP Project MR2733, Interim Report.

Palmsten, M, D Frank-Gilchrist, RC Mickey, A Penko, K Koetje, S Harrison, 2021. "Munition Mobility Hindcasts during Hurricane Sally," SERDP Project MR2733, SERDP/ESTCP Symposium 2021.

Poelhekke L, WS Jäger, A van Dongeren, TA Plomaritis, R McCall, Ó Ferreira, 2016. "Predicting coastal hazards for sandy coasts with a Bayesian Network," Coastal Engineering 118, pp. 21-34.

Rennie SE, 2022. Development of a Decision Tool: Underwater Munitions Expert System, Johns Hopkins University/Applied Physics Laboratory, FPS-R-22-*in preparation*, SERDP Final Report for MR19-1126, March 2022.

Rennie SE, A Brandt, 2017. "Status of Underwater Impact Penetration Modeling for use in the Underwater Munitions Expert System," Johns Hopkins University/Applied Physics Laboratory, FPS-T-17-0456, SERDP Interim Report for MR-2645, November 2017.

Rennie, SE, A Brandt, 2019. "Improved Models for Burial, Exposure, and Migration of Underwater Munitions", SERDP Project Interim Report MR-2645, JHU/APL Technical Report, FPS-T-19-0332.

Rennie, SE, A Brandt, 2020. “Community Feedback and Design of Visualization Output for the Underwater Munitions Expert System,” Johns Hopkins University/Applied Physics Laboratory, FPS-R-20-0281, SERDP Interim Report MR19-1126, July 2020

Rennie, SE, A Brandt, 2021. “Object Burial by High-Energy Forcing: Liquefaction and Granular Sorting”, SERDP Project Interim Report MR19-1126, JHU/APL Technical Report, FPS-T-20-0685.

Rennie, SE, A Brandt, CT Friedrichs, CT, 2017. “Initiation of motion and scour burial of objects underwater”, *Ocean Engineering, 131*, 282–294. dx.doi.org/10.1016/j.oceaneng.2016.12.029.

Rennie SE, A Brandt, JG Ligo, 2019. “Prototype Underwater Munitions Expert System: Demonstration and User's Guide,” Johns Hopkins University/Applied Physics Laboratory, FPS-R-19-0695, SERDP Guidance Document, MR-2645, November 2019.

Soulsby, R, 1997. Dynamics of Marine Sands, London: Thomas Telford.

Teichman, JA, J Macheret, SM Cazares, 2017. *UXO Burial Prediction Fidelity*, Institute for Defense Analyses, IDA NS D-8616.

Traykovski, P, T Austin, 2017. Continuous Monitoring of Mobility, Burial and Re-exposure of Underwater Munitions in Energetic Near-Shore Environments, SERDP Project MR-2319 Final Report.

Traykovski, P, Jaffre, F, 2020. Rapid Response Surveys of Mobility, Burial and Re-exposure of Underwater Munitions in Energetic Surf-Zone Environments and Object Monitoring Technology Development, SERDP Project MR-2729 Final Report.

Traykovski, P, Palmsten, M, 2020. Wave Resolved and Averaged Numerical Modelling of UXO Migration, SERDP Proposal MR21-S1-1341.

Young, CW, 1997, “Penetration Equations”, Sandia National Laboratory Contractor Report, SAND97-2426. p. 24.

Appendices

A.1 Netica for UnMES : Introduction

Bayesian Networks are a method of representing a joint probability distribution between a set of random variables by representing dependencies of variables in a graphical structure. They are often used in machine learning applications, as well as in the construction of expert systems. A Bayesian Network represents the relationship between several variables as a directed acyclic graph (in the Graph Theory sense), such that the joint distribution of the random variables factors as a product of the probability distributions of the individual variables conditioned on their parents in the graph. The Bayesian Network representation of a distribution is not unique, and does not necessarily imply any causality between the variables. Nevertheless, Bayesian Networks provide a useful mathematical tool to describe probability distributions with structured relationships between variables, which we use in the Underwater Munitions Expert System (UnMES) to encode our knowledge on underwater munition behavior based on first-order physics models. A detailed overview of Bayesian Networks can be found in several standard textbooks [1] and is beyond the scope of this memo.

This memo documents some tools used to manipulate Bayesian Networks in MATLAB in order to work with UnMES [2]. The tools in this memo can be used to read in Bayesian Networks into Matlab, train them, and export the resultant Bayesian Networks to files which can be ingested later by Norsys Netica or with the MATLAB tools provided here.

A comparison of various packages for graphical models (of which Bayesian Networks are one variety) is available at `https://www.cs.ubc.ca/~murphyk/Software/bnsoft.html`. Netica operates by discretizing continuous variables and implementing inference on discretized variables. It is a package that has been used on several JHU/APL projects, and has a reasonable licensing cost for both the application and the Application Programming Interface (API).

A.2 Setting Up Norsys Netica

Norsys Netica is a graphical user interface for manipulating and performing inference on Bayesian Networks. It currently only officially runs on Microsoft Windows platforms, though the manufacturer notes that users have made it work on Apple MacOS and Linux by use of compatibility layers (e.g. WINE `https://www.winehq.org/`). At JHU/APL, we generally do not use the compatibility layer approach as we have access to Microsoft Windows virtual machines through our centralized IT department. Some limited guidance is provided in the Netica online help (linked

[1] Barber, David. Bayesian reasoning and machine learning. Cambridge University Press, 2012.

Murphy, Kevin P. Machine learning: a probabilistic perspective. MIT press, 2012

[2] See, for example, Rennie et al [2019]

below) for MacOS and Linux. In general, Netica has been more stable running under Windows.

The Bayesian Network software Netica is available from Norsys Software at

https://www.norsys.com/netica.html

By installing and using Netica, you are agreeing to the terms of use, set forth in the license agreement, which is provided during the software download from Norsys in a separate document. Note that an initial exploration of the Mode 1 (single site) UnMES Bayes Net (BN) may be performed without purchasing the Netica license in "limited mode", however network size is restricted. The license is relatively inexpensive.

Requirements: Netica Application requires a PC running any version of Microsoft Windows from XP to Windows10. The 64 bit version of Netica requires Windows 7 or higher. Installation requires less than 10 MB of hard disk space.

Netica will run well even on a very slow PC , using very little RAM (about 30 MB), but working with larger, complex Bayes Nets may require much more speed and large amounts of RAM. The prototype version of UnMES is small enough to not exceed this limit.

To run Netica from a Mac computer, you must install a compatibility layer for running Windows programs. We generally use a Windows virtual machine accessed through Microsoft Remote Desktop.

Windows Installation:

1. First obtain the Netica package file. Download it from the Norsys website. The name of the file will beNetica_Win.exe.
2. Choose “Run” from the download dialog box, or save the file to disk and then double-click its icon.
3. When a dialog box appears, enter where on your hard disk you want the Netica folder placed. You can put it anywhere you wish; popular choices areC:\Netica orC:\Program Files\Netica. Make sure there is a drive letter (e.g. “C:\ ”) , or at least a slash ("\") at the front of the location. You don’t have to include version information in the name, because a folder called Netica ### will be created within it, where ### is the version number.
4. Click the UnZip button and then close the dialog box.

Running:

The UnMES software files have the file extension .neta or .dne, and are linked to the Netica executable Netica.exe (64-bit version) which resides in the Netica ### *home folder* which is saved by the installation procedure, and has the icon  . Within the home folder will be all the files required for Netica, including the 32-bit version,Netica32.exe, The first time you run an

executable, you should right-click on it, and choose "Run as administrator". If you want you can run it by double-clicking it, but at some time in the future you should run it as administrator to fully register it with the system (for proper icons, opening document by clicking it, COM interface, etc.). On some systems it may take a long time to start the first time; the next time it will start quickly.

If you wish Netica to be on your Start menu, you can use the normal Windows method of dragging the Netica.exe icon  from the Netica ### folder (as mentioned above) and dropping it on the Start button. Or drag it to the desktop if you want to make a shortcut there. During your exploration of UnMES, you may generate multiple versions of the .neta files. It is best to keep these files in a separate working folder, rather than the home folder. Note that the.neta and .dne files have the Windows icon  .

Password:

The enter-password dialog box will appear, so when you obtain a license *password* from Norsys by e-mail or on your invoice, you may type it in, or better yet copy and paste it in. If you later wish to remove or change the password, choose **File → Netica Password** from Netica's menu.

Tutorial and Help:

With Netica installed and running, you can get an introduction to how it works by doing the operations described in the Quick Tour, or check out the New Features. The online help is available at `https://www.norsys.com/WebHelp/NETICA.htm`. It is recommended that the user complete the "Quick Tour" included with Netica before proceeding.

A.3 Matlab-Netica Interface

In this section, we describe a few MATLAB scripts/functions to manipulate Bayesian Networks from MATLAB via the Netica-J API. They have been tested on Microsoft Windows and MacOS. Throughout this section, we will refer to the MATLAB startup directory (MSD) to the directory MATLAB starts up in. One can find this by starting MATLAB and using the "pwd" command to get the startup directory. Typically for Windows users this will be "`C:\Users\(UserName)\Documents\MATLAB`" and, for MacOS users, "`/Users/(UserName)/Documents/MATLAB`". The MSD can be set in MATLAB by entering the Preferences pane and selecting General, and specifying the "Initial Working Folder". It is recommended that the MSD is hard coded (i.e. MATLAB's Inital Working Folder is specified to be a fixed folder rather than the last used folder). It is recommended to use the typical directories listed above (which are the MATLAB defaults).

A3.1 Setting up Netica-J

This sub-section contains details which are operating system specific. One should download the Netica-J API from `https://www.norsys.com/netica-j.html` for their operating system and extract it into the MSD. The Netica-J API allows for manipulating Bayesian Networks in the Netica format from the Java programming language. MATLAB provides simple interfaces

to call Java code, allowing manipulation of Bayesian Networks in the Netica format from MATLAB. If one wishes to modify or extend the provided MATLAB code, it is highly suggested that the user have a copy of the Netica-J user manual and javadocs (available at the download site) and is familiar with the MATLAB-Java interfaces `https://www.mathworks.com/help/matlab/using-java-libraries-in-matlab.html`.

For Windows users, the MSD will now contain:

1. Netica.dll
2. NeticaJ.dll
3. NeticaJ.jar
4. NeticaJ.lib

whereas for MacOS users, the MSD will now contain:

1. libnetica.a
2. libNeticaJ.jnilib
3. NeticaJ.jar

Once this has been setup, one needs to make two files in the MSD so that MATLAB will be able to load the Netica-J library. The first is **javaclasspath.txt**, which should contain

```
.
./NeticaJ.jar
```

The second is **javalibrarypath.txt**, which on Windows should contain

```
.
```

and on MacOS contain

```
.
./libNeticaJ.jnilib
```

A.3.2 Core MATLAB interface

The core MATLAB Netica interface used in the development of UnMES consists of several files:

1. Netica_Init.m : Used to initalize Netica-J API in MATLAB (Sec. B.1)
2. Netica_load.m : Loads a Bayesian Network that is compatible with Netica (.dne/.neta) (Sec.B.2)
3. Netica_train.m : Given cases of data (case files), train a Bayesian Network using Netica-J (Sec. B.4)
4. Netica_save.m : Saves a Bayesian Network to a .dne/.neta file (Sec. B.3)

The MATLAB Netica interface files should be placed in the MSD. Throughout, "»" denotes the MATLAB interpreter; these commands can also be integrated in a MATLAB script. It is assumed that the working directory is always the MSD.

Our typical MATLAB workflow is to load a Bayesian Network (usually designed in the Netica application), run Monte Carlo simulations of the relevant physics (not documented in this memo), train the Bayesian Network based on the Monte Carlo simulations and save the resultant model for further manipulation, either in MATLAB or the Netica application.

A.3.2.1 Initialization

Before the use of any Bayesian Networks in MATLAB, the Netica_Init.m script should be run.

```
>> import norsys.netica.*;
>> Netica_Init
```

No output will be produced on successful execution. If errors occur related to JAVA, verify that the setup instructions in Section 3.1 have been performed correctly. Once the Netica_Init.m script has been run, other Netica-J functions can be used as they would in a native Java application.

A.3.2.2 Loading a Bayesian Network

To load a Bayesian Network into MATLAB, one can use the Netica_load.m script:

```
>> bn= Netica_load('test.neta'); % Loads test.neta
```

The variable bn will contain a norsys.netica.Net Java object representing the Bayesian Network in test.neta at this point. It can be manipulated using the functions of the Netica-J API in Matlab. Typically, the network being loaded was either one that has been manipulated in MATLAB before or one designed in the Netica application.

A.3.2.3 Training a Bayesian Network

One can train the Bayes Network with a "case file" (defined below) containing cases of data (e.g. simulated physics which we wish the Bayesian Network represents) by

```
>> bn_out = Netica_train(bn,["casefile1.txt","casefile2.txt",...], cpte );
% Trains the Bayes net bn with the data in casefile1.txt, casefile2.txt,... ;
% if cpte=1, reset the conditional probability tables before training
```

The Netica_train function takes in a Bayesian network loaded into matlab by Netica_load (in the previous subsection as a norsys.netica.Net object), as well as a string array of case files containg data to train the Bayesian Network. It also takes in a third parameter, which is 1 if previously learned information in the Bayesian network is to be cleared. Any other value of the third parameter retains the previous trained information. Typically, cpte=1 is not used, as the loaded Bayesian Network usually has not been trained and has been saved from the Netica interface after being designed.

For our intents and purposes, a case file is an ASCII text file where the first line consists of names of variables in the Bayesian network delimited by tabs. Subsequent lines consist of cases. A case is a set of values for the variables in the Bayesian Network that occur together, e.g. one set of envirnomental conditions and corresponding migration for a munition. The full description of case files is given in `https://www.norsys.com/WebHelp/NETICA/X_Case_File_Format.htm`. An example case file is given below.

```
Hsig        T        Uc        alpha        beta        Um        IB        dHsdt        TB
0.7         7.7         0.15          10          5          0.49         0.95         0.08          0.95
0.7         3.4         0.42          10          5          0.31         0.69         0.56          0.71
...
```

In this case, there are variables Hsig, T, Uc, alpha, beta, Um, IB, dHsdt, TB. The first case consists of a data point where Hsig=0.7, T=7.7, ..., while the second case consists of a data point where Hsig=0.7, T=3.4, ...

Each variable is separated by a tab. Not all variables need to be present in the case file; for example, the first casefile may contain data related to a few variables and their parents, while the next may contain data on other variables and their parents rather than the more complex task of specifying all variables in the network jointly. For example, a case file may relate significant wave height, current, period, ... to bottom orbital velocity to fill in the conditional probability table for bottom orbital velocity, whereas another case file may relate bottom orbital velocity, initial burial, ... to the total burial a munition experiences.

A.3.2.4 Saving a Bayesian Network

Once a Bayesian Network has been manipulated through MATLAB, one often wants to save it for further use. This can be done with the Netica_save function which takes in a Bayesian Network from the Netica_load function or Netica_train function (a norsys.netica.Net object) and an output filename (ending in ".dne" or ".neta").

```
>> Netica_save(bn,'output.neta')
```

Upon completion of this call, output.neta contains the Bayesian Network in bn, and is ready for inference (e.g. in MATLAB or through the Netica application). The .dne file format is text based, and is supported by several other Bayesian Network packages other than Netica. The .neta file format is a binary format, which is smaller and for use in Netica.

A.3.2.5 Complete Example

Below is a minimal complete example of a MATLAB script integrating the steps above. It uses an ancillary script to get a list of case files (data) described in Sec. C.1.

```
% Test the Netica read and Write functions.
% Assumes our data is in a folder called "Data".

import norsys.netica.*;
Netica_Init % Loads netica license

% Load the model in DemoBN.neta
bn=Netica_load('DemoBN.neta');

%Get list of data cases
training_data=getListOfCases('Data');
```

```
12
13 %Train the model (and clear all CPT tables before training)
14 bn_trained=Netica_train(bn,training_data,1);
15
16 %Save trained model to test.neta
17 Netica_save(bn_trained,'test.neta')
```

Ancilliary functions are described in Appendix C.

JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

Appendix B Core Matlab - Netica Code

This section provides source code listings for the MATLAB functions used to manipulate Bayesian Networks.

B.1 Netica_init.m

```
% Netica_Init
% Run this before using any Netica functions



% Your Netica password
NeticaPWD='(insert your Netica API password here)';
% '' also works as a password (for small networks)


import norsys.netica.*; % Allows use of Netica-J API


% The remaining code simply constructs the background environment for
% Netica to run.
% See the relevant Javadoc for details:
% https://www.norsys.com/netica-j/docs/javadocs/norsys/netica/Environ.html
global Neticaenv

if (~exist('Neticaenv') || isempty(Neticaenv))
    Neticaenv=Environ.getDefaultEnviron;
    Neticaenv=Environ(NeticaPWD);
end

% We use NETICA_CMD in some of our GUI Tools
% that are Windows only
global NETICA_CMD
NETICA_CMD='"C:\Netica\Netica 604\Netica.exe"';
           % The double quotes are necessary due to the space
```

B.2 Netica_load.m

```
% Netica_load  Load a Bayes Net from Netica (dne or neta format)
%               and compiles it.
%
%        Input:
%                (filename) String containing file name of Netica
   dne/neta
%                file to be loaded
%        Output:
%                 (bn) Object containing Bayes Net as Netica-J Net
    object
%                      i.e. Object containing Bayes Net
%
% Usage: bn=Netica_load(filename)


function bn=Netica_load(filename)

    if (exist(filename,'file')==2) % Check if network file exists
        import norsys.netica.*;  % Allows use of Netica-J API
        streamer=Streamer(filename); % Open filename for reading
        bn=Net(streamer); % Read  Bayes net from filename, make
           Net object
        bn.compile(); % Compile Bayes Net (as in Netica)
        streamer.finalize(); % Close filename for reading
    else
        error( ['File Not Found: ' filename] );
    end
end
```

B.3 Netica_save.m

```
% Netica_save  Saves a Bayes Net from Netica (dne or neta format)
%              and compiles it.
%
%        Input:
%                (bn) Object containing Bayes Net (as a Netica-J Net object)
%                     Such an object is produced by Netica_load, Netica_train
%                (filename) String containing filename to save Bayes net
%                           into, in Netica format. Should end with .neta or
%                           .dne. Caller is responsible for ensuring correct
%                           file ending.
%
%        Output: Does not return anything. The Bayes Net in bn should be
%        saved in Netica format into filename, but there is no way to check
%        if the save succeeded from Java/Matlab.
%
% Usage: Netica_save(bn,filename)



function Netica_save(bn,filename)
      import norsys.netica.*; % Allows use of Netica-J API
      streamer=Streamer(filename); % Open filename for writing
      bn.write(streamer); % Write Bayes Net bn to filename.
      % This looks like bad practice (not checking a return value)
      % But it seems like Netica doesn't return stuff if the write fails.
      streamer.finalize(); % Close filename for writing
end
```

B.4 Netica_train.m

```
% Netica_train  Takes a list of case files and uses Netica to train the
%               the Bayes net bn via the counting algorithm (i.e. populate
%               the conditional probability tables (CPT)).
%
%               Input:
%                       (bn) Bayes Net as a Netica-J Bayes Net object
%                            Produced by Netica_load or previous runs of
%                            Netica_train.
%
%                       (files) List of file names (as a String Array)
%                               of cases files for training the Bayes Net in
%                               bn. Can be produced by getListOfCases.
%
%                               A string array can be inputted as
%                               ["filea.txt","fileb.txt",...]. Note the
%                               double quotes. Requires Matlab R2016b+.
%
%                               Files must be in a flat file
%                               format which Netica can read cases. See
%                               https://www.norsys.com/WebHelp/NETICA/
%                               X_Learning_From_a_Case_File.htm
%                               for details on required formatting. The
%                               files do not need to be homogeneous; for
%                               example, one can have some Excel files, CSV
```

```
25 %                                files, and text files with
   appropriate
26 %                                formatting.
27 %
28 %                                No input validation is performed
   on the
29 %                                files.
30 %
31 %                          (cpte) If this is set to 1, the CPT &
   Experience for
32 %                                learning are set to zero for each
   node in the
33 %                                Bayes Net. If cpte is any value
   that is not
34 %                                1, then the CPT and Experience is
   untouched
35 %                                prior to training. Further details
   on these
36 %                                terms is provided on the Netica
37 %                                documentation:
38 %                                https://www.norsys.com/WebHelp/
   NETICA/
39 %
   X_Counting_Learning_Algorithm.htm
40 %                 Output:
41 %                          (bn) Bayes Net as a Netica-J Bayes Net
   object
42 %                               Produced by taking the input bn, and
   training
43 %                               with the list of casefiles in files.
44 %
45 %
46 % Usage: bn=Netica_train(bn,files,cpte)
47 %
48
49 function bn=Netica_train(bn,files,cpte)
50     import norsys.netica.*; % Allows use of Netica-J API
51     if (cpte==1) % If we want to clear the CPT tables
52         nodelist=bn.getNodes(); % Get a list of nodes in Bayes
              Net
```

```
53          for i=0:(nodelist.size()-1) % Java uses zero based
               indexing
54              nodelist.get(i).deleteTables(); % Delete CPT Table
                   for each
55                                                % node
56          end
57      end
58
59      for i=1:length(files) % For each input file of cases
60          strm=Streamer(files(i)); % Open case file for reading
61          bn.reviseCPTsByCaseFile(strm,[],1.0); % Update CPT by
               counting
62                                                  % algorithm using
                                                       cases in
63                                                  % current case file
64                                                  % Note: [] = null
                                                       in Java
65          strm.finalize(); % Close case file for reading
66      end
67  end
```

Appendix C Ancillary MATLAB Code for Netica Manipulation

This section provides some useful MATLAB utilities that can aid in training Bayesian Networks.

C.1 getListOfCases.m

This Aunction Aets A list of All Ales An A Aiven directory And Aeturns Ahem As A Atring array. At As often Ased Ao Aenerate Ahe list of case Ales Ao Ae Aassed Ao Aetica_train (Sec. B.4).

```
% getListOfCases  Generates a list of case files suitable
%                 for passing to Netica_train
%
%               Input:
%                       (directory) String containing directory
   that
%                      contains only case files to be used for
   training in
%                      its top level (i.e. we do not look for
   case files
%                      recursively).
%
%               Output:
%                       (filenames) String array containing list
   of
%                      casefiles in directory.
%
% Usage: filenames=getListOfCases(directory)
%


function filenames=getListOfCases(directory)
    dirlist=dir(fullfile(directory)); % Get list of things
       contained in
                                      % directory
    subdr=[dirlist.isdir]; % Get list of subdirectories

    % Get list of filenames in directory which are not
       subdirectories as a
    % string array.
    filenames=string(fullfile(directory,{dirlist(~subdr).name}))
```

```
        .';
end
```

C.2 Netica_populateTablesfromEquations.m

Sometimes in a Bayesian Network, there are relationships between variables that can be described by an equation succienctly (e.g. given the direction of a munition and of the waves, we can compute the angle between the munition and the waves using simple trigonometry). In cases where relationships can be described by an equation , Netica can implement those relationships by sampling inputs and outputs of the equation automatically, versus having to generate case files for those relationships and passing them to Netica_train.m; this can save significant training time due to the cost of file reading.

The code is given below.

```
% Netica_populateTablesfromEquations  Takes a bn and a list of nodes with
%               equations and populates the relevant CPT's with those
%               equations.
%
%               Input:
%                       (bn) Bayes Net as a Netica-J Bayes Net object
%                            Produced by Netica_load or Netica_train.
%
%                       (equations) Cell array where first column is node
%                                   names and second column is corresponding
%                                   equations, specified as strings for the
%                                   Node.setEquation Netica-J API.
%
%                       (nsamp) Num. of samples to convert equation to cpt.
%               Output:
%                       (bn) Bayes Net as a Netica-J Bayes Net object
%                            Produced by taking the input bn, and
%                            implementing the equations.
%
%
% Usage: bn=Netica_populateTablesfromEquations(bn,equations,nsamp
```

```
    )
%

function bn=Netica_populateTablesfromEquations(bn,equations,nsamp
    )
     for i=1:size(equations,1)
         bn.getNode(equations{i,1}).setEquation(equations{i,2});
         bn.getNode(equations{i,1}).equationToTable(nsamp,true,
            false);
     end


end
```

The usage is simple; one provides a two column cell array where the first column contains the variable names and the second column contains the equations in the Node.setEquation Netica-J API (essentially, how Netica specifies equations) and number of data samples to generate to implement the equation. The code below implements two equations: beta (wDir, cDir) = acos(cos(pi/180*(wDir-cDir)))*180/pi and alpha (wDir, UXOdir) = acos(cos(pi/180*(wDir-UXOdir)))*180/pi using 1e6 samples in the Bayesian Network bn (e.g. generated from Netica_train or Netica_load)

```
>> eqns={'beta','beta (wDir, cDir) = acos(cos(pi/180*(wDir-cDir))
    )*180/pi';'alpha','alpha (wDir, UXOdir) = acos(cos(pi/180*(
    wDir-UXOdir)))*180/pi'};
>> trained_bn_with_eqns=Netica_populateTablesfromEquations(bn,
    eqns,1e6);
```

C.3 Retrieving a Belief Vector in MATLAB

Assuming one has a Netica Bayesian Network bn loaded into Matlab,

```
>> belief_vector = bn.getNode('SB').getBeliefs();
```

retrieves the belief vector of the node SB in the network bn. Further functions can be implemented via the norsys.netica.Node API (`https://www.norsys.com/netica-j/docs/javadocs/norsys/netica/Node.html`).

Currently, our UnMES GUI uses this functionality to retrieve beliefs in UnMES. The norsys.netica.Node API can be used to set beliefs as well, but as the user interface is under active research, we are still using the Netica application to modify beliefs at specific spatial areas.

Appendix D Direct Entry of Conditional Probability Table in Netica

The Netica software saves Bayesian Networks (BN) in two formats. The default format is in binary form with the file extension .neta. An alternative ascii text form is available with the file extension .dne or .dnet. This format can be edited with any text editor, and the contents of the file are fairly comprehensible to even a novice reader. Each node is described in a block of text prefaced by a line starting with the word "node", followed by the node name. All text blocks are terminated by the string "};".

Within the node block, a line with the string "parents=(); " designates that there are no parents, or inputs, to this node. The block for a child node, such as Bedform%Burial (name:BB) which has three parent nodes, UXO_type, Dune_Height, and Dune_Length, will have a line similar to "parents = (UXO, DuneHt, DuneL);". Note that the node names rather than titles are used. For clarity in the following discussion, a simple version of UnMES with few UXO types and an abbreviated set of Dune heights and length states is used.

Another set of lines defines the Probability Table for the node. Parent nodes, such as UXO_type (name: UXO), have a simple three line description, e.g.

```
probs =
      // bullet  7.0  mortar 3.0   howitz 2.6
        (0.3333333,   0.3333333,   0.3333333);
```

while a child node containing a Conditional Probability Table (CPT) has a longer description with one line for each combination of the parent nodes' states. An example of the CPT for child node Bedform%Burial(name:BB) is shown in Table D.1. The probabilities shown in this table were generated by a version of the Matlab code `script_BedformBurialPDT.m`, (Section 1.5), modified to use fewer dune states and only three UXO types in order to simplify the explanation. The code writes the CPT lines to a text file, which is then copied and pasted into the .dne file for PrototypeV2 UnMES. In Table D.1, the Netica-generated text is green, and text that is pasted in from the Matlab script output is black. Note that all text following the characters "//" in a line are comments, and Netica is not sensitive to their format.

In a step prior to editing the .dne file, the user must first force Netica to generate a CPT for the BB node. In the Netica GUI, the user selects the BB node and opens the properties dialog, then opens the Table dialog. When first building a BN, the Table entries will all be blank. Hand-entering a single line of Probabilities (see Figure D.1a), and then saving the BN out to .dne format, will cause Netica to build the full dummy table description (with the rest of the values set to undefined). Then it will be easier to see in the .dne file where the dummy table begins and ends. The text written out by `script_BedformBurialPDT.m` should be inserted, completing replacing the previous CPT values. Particular care must be taken to keep the exact format that Netica generated: the first line containing CPT values must start with a "(". The last line ends with ");", which precedes the comment section in that last line. Netica is not sensitive to indenting or spaces. After the text

generated by the Matlab script is pasted in, and Netica reads in the modified `.dne` file, the `BB` node Table dialog will look like Figure D.1b when viewed with the Netica GUI.

a)

Node: BB | Apply | OK | Reset | Close

Chance | Probability

UXO_type	Dune_Height	Dune_Length	lessthan20perc	perc20to50	perc50to75	perc75to100	fullyBuried
bullet 7.0	0.1 to 0.3	60 to 100					
bullet 7.0	0.1 to 0.3	morethan100					
bullet 7.0	0.3 to 0.7	15 to 30 m					
bullet 7.0	0.3 to 0.7	30 to 60 m					
bullet 7.0	0.3 to 0.7	60 to 100					
bullet 7.0	0.3 to 0.7	morethan100					
bullet 7.0	0.7 to 1.1	15 to 30 m					
bullet 7.0	0.7 to 1.1	30 to 60 m	0	0	0	0	1
bullet 7.0	0.7 to 1.1	60 to 100					
bullet 7.0	0.7 to 1.1	morethan100					
mortar 3.0	0 to 0.1	15 to 30 m					
mortar 3.0	0 to 0.1	30 to 60 m					
mortar 3.0	0 to 0.1	60 to 100					

b)

BB Table (in Bayes net ExampleShortBB)

Node: BB | Apply | OK | Reset | Close

Chance | Probability

UXO_type	Dune_Height	Dune_Length	lessthan20perc	perc20to50	perc50to75	perc75to100	fullyBuried
bullet 7.0	0.1 to 0.3	60 to 100	.288339	.0273716	.0227472	.0228721	.63867
bullet 7.0	0.1 to 0.3	morethan100	.619156	.0146478	.0121731	.01224	.341783
bullet 7.0	0.3 to 0.7	15 to 30 m	.0102177	.0151044	.0124389	.0128832	.949356
bullet 7.0	0.3 to 0.7	30 to 60 m	.01022	.0151078	.0122195	.0126639	.949789
bullet 7.0	0.3 to 0.7	60 to 100	.27734	.0109986	.00912387	.00912387	.693413
bullet 7.0	0.3 to 0.7	morethan100	.61327	.00588589	.00488262	.00488262	.371079
bullet 7.0	0.7 to 1.1	15 to 30 m	.00621946	.00799645	.00710795	.00666371	.972012
bullet 7.0	0.7 to 1.1	30 to 60 m	.00577649	.00844257	.00688736	.00688736	.972006
bullet 7.0	0.7 to 1.1	60 to 100	.274216	.00599925	.00512436	.00499938	.709661
bullet 7.0	0.7 to 1.1	morethan100	.611598	.00321049	.00274229	.00267541	.379774
mortar 3.0	0 to 0.1	15 to 30 m	.3243	.485562	.190138	0	0
mortar 3.0	0 to 0.1	30 to 60 m	.32415	.485892	.189958	0	0
mortar 3.0	0 to 0.1	60 to 100	.506687	.354581	.138733	0	0

Figure D.1 a) single line of probabilities is entered in `BB`'s table using the Netica GUI. b) section of the table for node `BB` after pasting in Matlab-generated CPT lines into the `.dne` representation.

Table D.1 Conditional Probability Table for (simplified) UnMES node Bedform%Burial (BB)

```
node BB {
   discrete = FALSE;
   states = (lessthan20perc, perc20to50, perc50to75, perc75to100, fullyBuried);
   levels = (0, 0.2, 0.5, 0.75, 1, 5);
   kind = NATURE;
   chance = CHANCE;
   parents = (UXO, DuneHt, DuneL);
   probs =
      // lessthan20perc perc20to50 perc50to75  perc75to100  fullyBuried  // UXO    DuneHt     DuneL
      (1.00844e-01 , 1.49711e-01 , 1.24833e-01 , 1.25278e-01 , 4.99334e-01 ,  // bullet 0.00 to 0.10 15 to 30
      1.00422e-01 , 1.49745e-01 , 1.25083e-01 , 1.24861e-01 , 4.99889e-01 ,   // bullet 0.00 to 0.10 30 to 60
      3.43082e-01 , 1.09486e-01 , 9.12386e-02 , 9.12386e-02 , 3.64954e-01 ,   // bullet 0.00 to 0.10 60 to 100
      6.48452e-01 , 5.85914e-02 , 4.88262e-02 , 4.88262e-02 , 1.95305e-01 ,   // bullet 0.00 to 0.10 more than 100
      2.53221e-02 , 3.77610e-02 , 3.10973e-02 , 3.10973e-02 , 8.74722e-01 ,   // bullet 0.10 to 0.30 15 to 30
      2.53277e-02 , 3.75472e-02 , 3.08820e-02 , 3.15485e-02 , 8.74695e-01 ,   // bullet 0.10 to 0.30 30 to 60
      2.88339e-01 , 2.73716e-02 , 2.27472e-02 , 2.28721e-02 , 6.38670e-01 ,   // bullet 0.10 to 0.30 60 to 100
      6.19156e-01 , 1.46478e-02 , 1.21731e-02 , 1.22400e-02 , 3.41783e-01 ,   // bullet 0.10 to 0.30 more than 100
      1.02177e-02 , 1.51044e-02 , 1.24389e-02 , 1.28832e-02 , 9.49356e-01 ,   // bullet 0.30 to 0.70 15 to 30
      1.02200e-02 , 1.51078e-02 , 1.22195e-02 , 1.26639e-02 , 9.49789e-01 ,   // bullet 0.30 to 0.70 30 to 60
      2.77340e-01 , 1.09986e-02 , 9.12386e-03 , 9.12386e-03 , 6.93413e-01 ,   // bullet 0.30 to 0.70 60 to 100
      6.13270e-01 , 5.88589e-03 , 4.88262e-03 , 4.88262e-03 , 3.71079e-01 ,   // bullet 0.30 to 0.70 more than 100
      6.21946e-03 , 7.99645e-03 , 7.10795e-03 , 6.66371e-03 , 9.72012e-01 ,   // bullet 0.70 to 1.10 15 to 30
      5.77649e-03 , 8.44257e-03 , 6.88736e-03 , 6.88736e-03 , 9.72006e-01 ,   // bullet 0.70 to 1.10 30 to 60
      2.74216e-01 , 5.99925e-03 , 5.12436e-03 , 4.99938e-03 , 7.09661e-01 ,   // bullet 0.70 to 1.10 60 to 100
      6.11598e-01 , 3.21049e-03 , 2.74229e-03 , 2.67541e-03 , 3.79774e-01 ,   // bullet 0.70 to 1.10 more than 100
      3.24300e-01 , 4.85562e-01 , 1.90138e-01 , 0.00000e+00 , 0.00000e+00 ,   // mortar 0.00 to 0.10 15 to 30
      3.24150e-01 , 4.85892e-01 , 1.89958e-01 , 0.00000e+00 , 0.00000e+00 ,   // mortar 0.00 to 0.10 30 to 60
      5.06687e-01 , 3.54581e-01 , 1.38733e-01 , 0.00000e+00 , 0.00000e+00 ,   // mortar 0.00 to 0.10 60 to 100
      7.36004e-01 , 1.89753e-01 , 7.42425e-02 , 0.00000e+00 , 0.00000e+00 ,   // mortar 0.00 to 0.10 more than 100
      8.12972e-02 , 1.21279e-01 , 1.01288e-01 , 1.01288e-01 , 5.94847e-01 ,   // mortar 0.10 to 0.30 15 to 30
      8.13153e-02 , 1.21306e-01 , 1.01311e-01 , 1.01311e-01 , 5.94757e-01 ,   // mortar 0.10 to 0.30 30 to 60
      ... a number of lines are left out for brevity ...
      6.33603e-01 , 3.63186e-02 , 3.02990e-02 , 3.02321e-02 , 2.69547e-01 ,   // howitz 0.30 to 0.70 more than 100
      3.50955e-02 , 5.15327e-02 , 4.30920e-02 , 4.30920e-02 , 8.27188e-01 ,   // howitz 0.70 to 1.10 15 to 30
      3.46590e-02 , 5.15441e-02 , 4.31015e-02 , 4.31015e-02 , 8.27594e-01 ,   // howitz 0.70 to 1.10 30 to 60
      2.95213e-01 , 3.77453e-02 , 3.13711e-02 , 3.14961e-02 , 6.04174e-01 ,   // howitz 0.70 to 1.10 60 to 100
      6.22835e-01 , 2.01993e-02 , 1.67882e-02 , 1.68551e-02 , 3.23323e-01);   // howitz 0.7 to 1.1 morethan100;
```

Appendix E. Code Listings for Spatial Application Scripts

E.1 `make_network_input_D3D.m`

E.2 `UnMES_D3Dconnect.py`

make_network_input_D3D.m

Transform DELFT3D (D3D) simulation results for use with UnMES-D3D. Create Netica-compatible case file that includes spatial information.

```
% Original Code from M. Palmsten (USGS) & D. Frank-Gilchrist 2020
% Modifications and additional comments S. Rennie JHU/APL
% to  make Netica-style  "CASE" file for
% use with "D3D" version of UnMES which will take bottom flow velocity
 (Um) and direction (UmDIR)
% as environmental input instead of PeakWave Height (Hsig),
% Current Velocity (Uc),and  Wave and Current Directions (wDir and
 cDir)

% Version make_network_input_D3D+1ts shown here downselects to the
 single time step of interest
% Previous versions retained all time steps from Delft3D, but this
 script demonstrates the
% creation  of an UnMES CASE file of a single moment in time to be
 used an input to the
% Python module UnMES_D3Dconnect.py which applies the UnMES-D3D
 Bayesian Network to a
%  2D grid to create a spatial presentation of UnMES predictions based
 on Delft3D results.
close all
clear all

% load D3D data - Location depends on local disk organization %%%

%udir='G:\mpalmsten\Documents\projects\DUCK';
udir='/Users/rennise1/Documents/My Documents/MyNewWorkMac/SERDP/
PalmstenPenko/DuckDelft3D/Duck2015_2020xfers/';

ddir= udir;

ufil='DUCK2015_d3d_data';  % depends on the current Delft3D run of
 interest

load([ddir ufil '.mat']);

dsed = 0.26/1000;   % sediment grain size (d50) in meters
 representative of most of Duck FRF area
pref='frc';
flowmod=true;
wavemod=true;
debugplots = 0; % set to zero to avoid the quiver plots used to
 diagnosis the directional transformation

% set UXO Type
% These can be customized to match applied UnMES-D3D version
UXOFlag = 2;
```

```
if UXOFlag == 0
    UXOType = 2.5;%  (diameter in cm)
    UXOName = 'bullet';
elseif UXOFlag == 1
    UXOType = 12.7;
    UXTName = 'Naval5_38';
elseif UXOFlag == 2
    UXOType = 8.1;
    UXOName = 'mortar';
elseif UXOFlag == 3
    UXOType = 15.5;
    UXOName = 'howitzer';
elseif UXOFlag == 4
    UXOType = 12;
    UXOName = 'flare';
end
UXOType_m = UXOType/100;  % Diameter of UXO in meters

%outDir = 'C:\Users\mpalmsten\Documents\projects\SERDP\D3DCaseFiles';
outDir = '.';
% variable fil has been loaded in from the d3d_data results file
% and is a string describing this Delft3D run
outPath = [outDir,filesep,fil,'_',UXOName,'example.cas'];  % where the
 text CASE file will go


% grab times for current and wave model results
% Note: Code here assumes that tw is the important time designation
% and  that  tc and tw must be ~ the same
if flowmod
tc=vel.data.Time;
end
if wavemod
    if strcmp(region,'outer')
        tw=wave.Hs_out.data.Time;
    else
        tw=wave.Hs.data.Time;  % tests so far: region = 'inner-outer'
    end
end

% Removing NaNs at edges of grids
vel.data.X(end,:) = [];
vel.data.X(:,end) = [];

vel.data.Y(end,:) = [];
vel.data.Y(:,end) = [];

% current dir
vel_vec.data.X(end,:) = [];
vel_vec.data.X(:,end) = [];
vel_vec.data.X(1,:) = [];
vel_vec.data.X(:,1) = [];

vel_vec.data.Y(end,:) = [];
```

```
vel_vec.data.Y(:,end) = [];
vel_vec.data.Y(1,:) = [];
vel_vec.data.Y(:,1) = [];

vel_vec.data.XComp(:,end,:) = [];
vel_vec.data.XComp(:,:,end) = [];
vel_vec.data.XComp(:,1,:) = [];
vel_vec.data.XComp(:,:,1) = [];

vel_vec.data.YComp(:,end,:) = [];
vel_vec.data.YComp(:,:,end) = [];
vel_vec.data.YComp(:,1,:) = [];
vel_vec.data.YComp(:,:,1) = [];

Hs = permute(wave.Hs.data.Val, [2,3,1]);  % these are on a coarser
 grid than vel (FLOW grid)
Tp = permute(wave.Tp.data.Val, [2,3,1]);
Dir = permute(wave.Dir, [2,3,1]);         % Wave Dir Compass  =
 (relative to North)
Ub = permute(wave.Ub.data.Val, [2,3,1]);  % orbital velocity near
 bottom in m/s
XComp = permute(vel_vec.data.XComp,[2,3,1]);
YComp = permute(vel_vec.data.YComp,[2,3,1]);

 Ntime = numel(tw);
 itpeak = 786;  % select moment in time at  peak of storm on 2/11/2015
 for FRF   Index into tw
 % Custom Choice to CASE file of interest
 % Could have code here to scan Hs ( Wave Height ) to find largest
 time period.
 % Choice for itpeak time step determined outside of this code.

% bed level [change from inital level (m)]  Erosion/Accretion
EA = permute(eros.data.Val, [2,3,1]);
EA = EA*100; %  convert to cm (EA node in UnMES expects units of
 centimeters)
% Note that EA is on the same grid as the current vel.data.Val

% grid bathymetry depth     variable depth is input from d3d_data.mat
 on same grid as Lat & Lon
% Note : depth is not a structure
% Again, cut down by 1 to match grids in vel.data
h_m = depth(1:end-1,1:end-1);

% Preparation for Interpolation of waves onto higher resolution
 current grid
% Translate Lon & Lat degrees to FRF northing-easting coordinates
[ALat, ALon, spN, spE, YWave, XWave] = frfCoord(wave.Lon, wave.Lat); %
 get FRF coordinates for lower-resolution wave grid
[ALat, ALon, spN, spE, YAlong, XCross] = frfCoord(vel.data.X,
 vel.data.Y);  % this is to be the common hi-res grid
[ALat, ALon, spN, spE, YVel_vec, XVel_vec] = frfCoord(vel_vec.data.X,
 vel_vec.data.Y);
```

```
% Note:  to  fully match the (right-handed) UnMES coordinate system
 for Wave& Current ( & Um) Direction,
% the Y coordinates have to be sign-reversed

% (Clearly  downselection to just the time step of interest could be
 moved up here
% so it is not necessary to loop thru the full time series.
% But left for the moment in case there is a reason to process more
 than one time step)
for ii = 1:Ntime
    HsigIn = squeeze(Hs(:,:,ii));   % Significant Wave Height (m)
    HsigF = scatteredInterpolant(XWave(:),YWave(:),HsigIn(:));
    HsigHigh = HsigF(XCross(:),YAlong(:));
    HsigHigh = reshape(HsigHigh,size(XCross));
    Hsig(:,:,ii) = HsigHigh;
end

for ii = 1:Ntime                    % Wave Period (sec)
    TIn = squeeze(Tp(:,:,ii));
    TF = scatteredInterpolant(XWave(:),YWave(:),TIn(:));
    THigh = TF(XCross(:),YAlong(:));
    THigh = reshape(THigh,size(XCross));
    T(:,:,ii) = THigh;
end

for ii = 1:Ntime
    UbIn = squeeze(Ub(:,:,ii));           % bottom orbital velocity
    UbF = scatteredInterpolant(XWave(:),YWave(:),UbIn(:));
    UbHigh = UbF(XCross(:),YAlong(:));
    UbHigh = reshape(UbHigh,size(XCross));
    UbW(:,:,ii) = UbHigh;
end

for ii = 1:Ntime
    DirIn = squeeze(Dir(:,:,ii));        % Wave Direction Compass FROM
    DirF = scatteredInterpolant(XWave(:),YWave(:),DirIn(:));  % this
 is still compass direction but located at
    DirHigh = DirF(XCross(:),YAlong(:));                      % FRF
 Northing,Easting coordinate grid location
    DirHigh = reshape(DirHigh,size(XCross));
    waveDir(:,:,ii) = DirHigh;
end

for ii = 1:Ntime
    XDirIn = squeeze(XComp(:,:,ii));   % East-West component of
 Current
    XDirF = scatteredInterpolant(XVel_vec(:),YVel_vec(:),XDirIn(:));
    XDirHigh = XDirF(XCross(:),YAlong(:));
    XDirHigh = reshape(XDirHigh,size(XCross));
    XDir(:,:,ii) = XDirHigh;      % Easterly Current component -- will
 need to translate to Cross-shore
end

for ii = 1:Ntime
```

```
    YDirIn = squeeze(YComp(:,:,ii));   % North-South compnent of
 Current
    YDirF = scatteredInterpolant(XVel_vec(:),YVel_vec(:),YDirIn(:));
    YDirHigh = YDirF(XCross(:),YAlong(:));
    YDirHigh = reshape(YDirHigh,size(XCross));
    YDir(:,:,ii) = YDirHigh;       % Northerly Current component --
 need need to translate to Along-shore
end

% Rotate Current components so that they represent Cross and Along
 shore components rather
% than Easterly and Northerly
% The -18 deg changes from vectors in Lon,Lat space to a coordinate
 space aligned with the shore at Duck FRF.
%[Xcross, Yalong] = rot2d(Xc, Yc, -18);
q = -18 * pi/180;
XCrossVel = cos(q).*XDir - sin(q).*YDir;
YAlongVel = sin(q).*XDir + cos(q).*YDir;

% Organize data for UnMES "cases"

% current mag and dir
Uc= permute(vel.data.Val, [2,3,1]);  % Currents were already on hi-res
 grid   NOTE NOT USED in this code version
% Instead we work directly from current Components vel_vec.data.XComp
 & vel_vec.data.YComp
% note Uc represented the depth-averaged current.
% Consider h_m/2 to be  Z_ref

% Estimate near-bed velocity at height above seabed = Diameter of UXO
% similar to depth avg to near bottom conversion on D3D manual p203
 (Sec 9.4.1.1)
% corrected as per Donya Frank-Gilcrest's writeup UcB_Calcs 9/4/2020
kappa = 0.41;
z0 = 2.5*dsed/30;  % this is Soulsby DMS Eq 23c appropriate for
 hydrodynamically rough flow
halfwaterdepth = -0.5*h_m;
halfwaterdepth(halfwaterdepth<0.1) = NaN;
ustar = Uc .* kappa./(log(halfwaterdepth./z0));
UcB = ustar./kappa .* (log(UXOType_m./z0));   % total bottom current
 velocity is NOT Actually USED in this CODE
% The U & V components only are input to UnMES-D3D
ustarX = XCrossVel .* kappa./(log(halfwaterdepth./z0));
UcBx = ustarX./kappa * (log(UXOType_m./z0));
ustarY = YAlongVel .* kappa./(log(halfwaterdepth./z0));
UcBy = ustarY./kappa * (log(UXOType_m./z0));
% dcUcB = sqrt(UcBx.^2 + UcBy.^2);     doublecheck of calculations

if debugplots    % diagnostic for angular translation from compass to
 shore normal UnMES system.
    qskip = 9;    % to thin out quiver plots
    qscale = 1.2; % scaling for quiver arrows
    Xc = squeeze(XCrossVel(:,:,itpeak));   % select out field at  time
 step of interest
```

```
    Yc = squeeze(YAlongVel(:,:,itpeak));
    figure

 quiver(XCross(1:qskip:end,1:qskip:end),YAlong(1:qskip:end,1:qskip:end), ...
         Xc(1:qskip:end,1:qskip:end),Yc(1:qskip:end,1:qskip:end),
 qscale,'k');
      title('Current Direction  0 is Shore normal') ; axis equal
end

% rotate waveinto UnMES shoreline oriented coordinate system FROM
% where 0 deg is perpendicular (Shore Normal). Dir indicated FROM
% negative is from NE,  positive is from SE (right handed)

% For the  D3D version of UnMES called Prototype2022UnMES-
D3DV2<dsed>.neta it
% no longer passes Current Direction and Wave Direction (cDir and
 wDir) as Input,
% but more usefully provides DIRECTION of Um which will
% more accurately translate to Migration_Direction. Note this version
 does not use beta any more,
% and has removed both the Wave-Direction and Current_direction nodes.
% The Wave-UXO_Angle node ("alpha") is still retained but input PDF
 set
% to 50% angled and 50% perpendicular. Recommended to keep to that
 unless there is some strong
% reason (based on local knowledge) to change  -- See Sect 2.1
 Development of a Decision Tool:
% Underwater Munitions Expert System  Rennie, 2022, Final Report for
 MR19-1126.

waveDir = waveDir - 72;  % waveDir was From
  % wave.Dir which is provided from D3D in COMPASS direction;
  % after -72 deg this is now  aligned with shoreline at Duck FRF
ibad = waveDir < -90; % switch coordinates so the are -90 to 270 for
 new version of network so that wave directions are in same system as
 currents
waveDir(ibad) = waveDir(ibad) +360;

UbWx = -1*abs(UbW).*cosd(waveDir);  %  x component is cross-shore   -1
 Makes it FROM
UbWy = abs(UbW).*sind(waveDir);     %  y component is along-shore
if debugplots
    Xw = squeeze(UbWx(:,:,itpeak));   %  diagnostic plot for angular
 translation of Wave Direction
    Yw = squeeze(UbWy(:,:,itpeak));
    figure;

 quiver(XCross(1:qskip:end,1:qskip:end),YAlong(1:qskip:end,1:qskip:end), ...
         Xw(1:qskip:end,1:qskip:end),Yw(1:qskip:end,1:qskip:end),
 qscale,'b');
     title('Wave Direction  0 is Shore Normal (FROM)') ; axis equal
end
% angle between waves and currents  (beta) is  no longer used in
 UnMES-D3D because have the X,Y vector
```

```
% components of both wave and current contributions to form  total
 near bed velocity for UnMES.
% Previous formula  was Um = sqrt( UbW.^2 + UcB.^2 +
 2.*UbW.*UcB.*abs(cosd(beta))); which is Eq.7 in
% Friedrichs, Rennie & Brandt 2016 "Self-burial of Objects on Sandy
 Beds by Scour"

UmVecX = UbWx + UcBx;  % combined cross-shore flow component
UmVecY = UbWy + UcBy;  % combined along-shore flow component
Um = sqrt(UmVecX.^2 + UmVecY.^2);
UmDir = atan2(UmVecY,-1*UmVecX).*180./pi ;  %   note -1*Xcomponent
UmDir(UmDir < -135) = UmDir(UmDir < -135) + 360 ;  % Note:
 ProtypeUnMES-D3D UnMES uses angle convention:
    %  Left  -135 to -45          coming FROM the Left
    %  Offshore   -45 to +45      Note that 0 is Shorenormal (coming
 FROM offshore)
    %  Right +45 to +135
    %  Onshore  +135 to +225      coming FROM Onshore  going TOWARDS
 offshore

if debugplots
    figure;        %  diagnostic plot for Combined Wave + Current
 bottom flow velocity components
    Xm = UmVecX(:,:,itpeak);
    Ym = UmVecY(:,:,itpeak);

 quiver(XCross(1:qskip:end,1:qskip:end),YAlong(1:qskip:end,1:qskip:end), ...
          Xm(1:qskip:end,1:qskip:end),Ym(1:qskip:end,1:qskip:end),
 qscale,'b');
     title('Combined Direction UmDir (FROM)') ; axis equal
end

% Note Yalong and XCross
LatVelgrid = vel.data.Y; % Position of common grid in Latitude
LonVelgrid = vel.data.X; % and Longitude Degrees

%  Time Step of Interest repeated at each grid location for
 documentation of each case file line
tpeak = tw(itpeak) .* ones(size(LatVelgrid));

% make matrix for 1 UXO type repeated at each grid location
% Note: there might eventually be a source of information for varying
 UXO types throughout
% the sptail grid, in which case the values could be entered here
UXO = ones(size(LatVelgrid)).*UXOType;

Hsigit = Hsig(:,:,itpeak);
Tit = T(:,:,itpeak);

UmVecXit = UmVecX(:,:,itpeak);
UmVecYit = UmVecY(:,:,itpeak);
EAit = EA(:,:,itpeak);        % Note: this was already on full hires
 (vel_vec) current) grid  so did not need to be interpolated
```

```
ind = 1:length(Tit(:));  % index for each case file line
Umit = Um(:,:,itpeak);
UmDirit = UmDir(:,:,itpeak);

% Deal with bad values
% 1) get rid of values where bathymetry is too shallow
% 2) restrict values to limits in UnMES-D3D node bin range
ishal = find(h_m > -0.5); % must be at least 0.5 m water depth
    Hsigit(ishal) = 0.0;
    Umit(ishal) = 0.0;     UmDirit(ishal) = 0.0;
    UmVecXit(ishal) = 0.0; UmVecYit(ishal) = 0.0;
    EAit(ishal) = 0.0;
Tit(Tit>16 ) = 16;     %  D3D version of UnMES has Wave Period T from
 4 to 16 seconds
Tit(Tit<4 ) = 4;       %  Limit to allowable range rather han force to
 NaN
EAit(EAit > 30 ) = 30;      % limit to ranges currently in UnMES
 version Prototype201920_UnMESD3D_V3Fine.neta
EAit(EAit < -30 ) = -30;    % in cm
EAit(isnan(EAit)) = 0.0;    % still are some NaNs in EA? Unclear why

% collect all arrays ready to be printed out   This Version has
 locations denoted by Latitude and Longitude
dum=[ind; tpeak(:)'; LonVelgrid(:)'; LatVelgrid(:)';
 h_m(:)';Hsigit(:)'; Tit(:)'; Umit(:)'; UmDirit(:)'; UmVecXit(:)';
 UmVecYit(:)'; EAit(:)'; UXO(:)'];

fid =
 fopen([outPath(1:end-4),'_PeakStorm_LonLat',outPath(end-3:end)],'w');

% first output header line (see Netica CASE file format) that
 specifies the NODE name for each column
% Only the values to be input to  nodes  are REQUIRED by UnMES but in
 addition,
% we output Time, Location, and bathymetry for documentation and use
 by  visualization routines
fprintf(fid,'ind  t_datenum      lon       lat    h_m     Hsig     T
    Um     UmDir     UmX     UmY    EA    UXO \n');

% Write out full  array, one text line ( a "case") for each grid
 location
fprintf(fid,'%5.0f %10.4f %11.6f %11.6f %6.2f %5.2f %5.2f %5.2f %5.2f
 %6.3f %6.3f %7.2f %6.2f  \n',dum) ;
fclose(fid);

% If needed, the lines above could be repeated but with
% loactions designated by YAlong(:)', XCross(:)'
% instead of LonVelgrid(:)'; LatVelgrid(:)'
% so that visualizations using the FRF Northing
% and Easting Coordinate are easily created
```

Published with MATLAB® R2018b

```python
# -*- coding: utf-8 -*-
"""

@author: mpalmsten and sbateman  2019
Modifications S.E. Rennie  2020-2021
"""

#Load the Bayesian net.

import os
import gearth_module
from gearth_module import make_kml, gearth_fig
# Python MODULE for gearth downloaded from
# https://ocefpaf.github.io/python4oceanographers/blog/2014/03/10/gearth/
import matplotlib
from matplotlib import pyplot
import numpy
import pathlib
import pynetica
import scipy.io
import pandas
from simplekml import (Kml, OverlayXY, ScreenXY, Units, RotationXY,
                       AltitudeMode, Camera)
from palettable import colorbrewer


def checkError(env):
    '''
    Convenience function to raise a RuntimeError if there is a Netica error.
    '''

    report = pynetica.GetError_ns(env, pynetica.ERROR_ERR, None)
    if report is not None:
        mesg = pynetica.ErrorMessage_ns(report)
        raise RuntimeError(mesg)

# Create the Netica environment. This needs to be done once,
# before any other Netica API functions are called.
env = pynetica.NewNeticaEnviron_ns()
ret, mesg = pynetica.InitNetica2_bn(env)
if ret < 0:
    raise RuntimeError(mesg)

# Specify the UnMES-D3D Bayesian net to use for this Case file
net_file = pathlib.Path('./Prototype2022_UnMES-D3D_fine.neta')
stream = pynetica.NewFileStream_ns(str(net_file), env)
net = pynetica.ReadNet_bn(stream, pynetica.NO_VISUAL_INFO)
pynetica.DeleteStream_ns(stream)
checkError(env)
```

```
# Make sure the net is compiled (i.e. probabilties will update)
# and has no findings.
pynetica.CompileNet_bn(net)
pynetica.RetractNetFindings_bn(net)
checkError(env)

# Get a list of all the network nodes and their states.
class NodeInfo: pass
nodes_info = {}

all_nodes = pynetica.GetNetNodes2_bn(net)
num_nodes = pynetica.LengthNodeList_bn(all_nodes)
for i in range(num_nodes):
    node_info = NodeInfo()
    node_info.node = pynetica.NthNode_bn(all_nodes, i)
    node_info.name = pynetica.GetNodeName_bn(node_info.node)
    node_info.is_discrete = (pynetica.GetNodeType_bn(node_info.node) ==
pynetica.DISCRETE_TYPE)
    node_info.num_states = pynetica.GetNodeNumberStates_bn(node_info.node)

    if node_info.is_discrete:
        node_info.state_names = [
            pynetica.GetNodeStateName_bn(node_info.node, s)
            for s in range(node_info.num_states)
        ]
    else:
        node_info.levels = pynetica.GetNodeLevels_bn(node_info.node)

    nodes_info[node_info.name] = node_info

    if node_info.is_discrete:
        print(f'Node "{node_info.name}" is discrete with
{node_info.num_states} states:')
        for s in range(node_info.num_states):
            print(f'  "{node_info.state_names[s]}"')
    else:
        print(f'Node "{node_info.name}" is continuous with
{node_info.num_states} states:')
        for s in range(node_info.num_states):
            lo = node_info.levels[s]
            hi = node_info.levels[s+1]
            print(f'  {lo:g} to {hi:g}')

checkError(env)

# #####  Define Functions to query the net.  #####
# First, define a useful functions for querying the net for  a single case
def queryNetCase(input_vars_values, input_vars_names, output_vars_names):
    '''
```

```
    For a single case (i.e., a single value for each input variable),
    enter findings in the given nodes and return the probability
    distributions of the output nodes
    '''
    pynetica.RetractNetFindings_bn(net)

    for name, value in input_vars_values.items():
        if name in input_vars_names:
            # Added to handle missing input data ( * )
            # Do not update if input data is missing
            if value != '*':
                pynetica.EnterNodeValue_bn(nodes_info[name].node,
numpy.float64(value))
    output_vars_values = {
        name: pynetica.GetNodeBeliefs_bn(nodes_info[name].node)
        for name in output_vars_names
    }

    pynetica.RetractNetFindings_bn(net)

    checkError(env)

    return output_vars_values

# Define a the functions for querying the net for a series of cases
def queryNetCases(input_vars_cases, input_vars_names, output_vars_names):
    '''
    For a "timeseries" or a series of cases represting grid locations,
    query the net for each individual case and construct an output series.
    The output for each node will be a 2D array of probabilities
    indexed by [case,state] = the full probability distribution of
    the node for each case.
    '''
    # Broadcast the values so scalars can be treated like ndarrays.

    #input_vars_names = list(input_vars_cases)

    broadcasted = numpy.broadcast_arrays(*[
        input_vars_cases[name]
        for name in input_vars_names
    ])

    input_vars_cases_broadcasted = {
        input_vars_names[i]: b
        for i, b in enumerate(broadcasted)
    }

    num_cases = len(broadcasted[0])

    # Query the net for each case and build up the output case series.
```

```
    output_vars_cases = {}

    for c in range(num_cases):

        input_vars_values = {
            name: input_vars_cases_broadcasted[name][c]
            for name in input_vars_names
        }

        output_vars_values = queryNetCase(input_vars_values,
input_vars_names, output_vars_names)

        for name, probs in output_vars_values.items():

            if name not in output_vars_cases:
                output_vars_cases[name] = numpy.empty(
                    shape = (num_cases, len(probs)),
                    dtype = numpy.float32
                )

            output_vars_cases[name][c,:] = probs

    return output_vars_cases
# #####  END Definition of Functions to query the net.  #####


#  Specify  where the CASE file to process is
case_path_pre = './D3DCaseFiles/'
case_name = 'DuckExampleD3D.txt'
case_path = case_path_pre+case_name

do_kmz_plots = 0  # for quick debug runs, do not bother with KMZ output
# define grid size and special  location of interest indices into the grid.
# The following values hard-coded for Duck FRF Delft3D grid
nxgrid = 342    # this is Latitude ( alongshore)  dx = dlat = 2.8e-5
nygrid = 146    # this is cross shore (Longitude) dy = dlon = 1.06e-4
mx = 109   # note MATLAB is grid(146,342) while python is grid[342,146]
my = 74    # location of Calantoni Duck2015 observ for model-data compare

out_path_pre = './UnMES_pyNeticaResults/'
out_path_add = 'DuckExample_Howitz/'
out_path = out_path_pre + out_path_add

# create directory to output overlay images and kmz file
if os.path.exists(out_path)==0:
    try:
        os.makedirs(out_path)
    except OSError:
        print('Directory creation failed.')
```

```
        print( Directory creation failed! )
    else:
        print('Sucessfully created directory')

# Read in the case file and make  into a dictionary
caseData = pandas.read_csv(case_path,sep = '\s+')
input_vars_cases = caseData.to_dict('list')

##  MODIFICATIONS possible manipulation of UnMES input values at this point

#  Could add input value for initial burial here
IBvec = numpy.array(input_vars_cases['UXO'])*0.0 + 0.1
input_vars_cases['IB'] = IBvec  # add new column (key) for Initial Burial

#  Could modify the choice of UXO_Type at this point
UXOvec = numpy.array(input_vars_cases['UXO'])  # was set to Mortar before
UXOvec[:] = 15.5  #  set entire field to contain ALL Howitzer
#UXOvec[:] = 12  # or set entire field to contain ALL FLares
input_vars_cases['UXO'] = UXOvec

# OUTPUT (Predicted) variable names to make plots of
output_vars_names = {
    'TB':'Fraction Burial (-)', 'Mdist':'Migration Distance (m)',
    'Mdir':'Migration Direction (deg)',  'FB':'Final Burial'
    }

# input variable index into Netica
input_vars_names = [
      'IB', 'Um', 'T','UmDir',  'UXO',  'EA'
    ]
# Choose colormaps for plots
input_cmaps = ['Greens','Purples','Greens','Reds','Reds', 'greys']
output_cmaps = ['YlOrRd','YlGnBu','PuBuGn','YlOrRd']

#  Do UnMES predictions for all grid locations  *****
output_vars_cases = queryNetCases(input_vars_cases, input_vars_names,
output_vars_names.keys())

# Plot inputs to the model
lat_grid = numpy.array(input_vars_cases['lat']).reshape(nxgrid,nygrid)
lon_grid = numpy.array(input_vars_cases['lon']).reshape(nxgrid,nygrid)
pixels = 1024 * 10  # plot resolution

for var in input_vars_names:
    in_grid = numpy.array(input_vars_cases[var]).reshape(nxgrid,nygrid)
    # change missing data * to nan for plotting
    in_grid = numpy.where(in_grid=='*', numpy.nan, in_grid)
    # make floating point instead of strings in case of *
    in_grid = in_grid.astype(numpy.float)

    # check if all one value (do not bother to plot)
```

```
    # check if all one value (do not bother to plot)
    if numpy.nanmin(in_grid) != numpy.nanmax(in_grid):
       fig, ax =
gearth_fig(llcrnrlon=numpy.array(min(input_vars_cases['lon'])),
                   llcrnrlat=numpy.array(min(input_vars_cases['lat'])),
                   urcrnrlon=numpy.array(max(input_vars_cases['lon'])),
                   urcrnrlat=numpy.array(max(input_vars_cases['lat'])),
                   pixels=pixels)
       cmap = colorbrewer.get_map(input_cmaps[input_vars_names.index(var)],
'sequential', 5, reverse=False).mpl_colormap
       if var == 'Hsig' or var == 'T'  or var == 'UXO'  or var == 'Um'  :
         cs = ax.pcolormesh(lon_grid, lat_grid, in_grid, cmap=cmap,
vmin=numpy.nanmin(in_grid), vmax=numpy.nanmax(in_grid))
       elif var == 'IB':
         cs = ax.pcolormesh(lon_grid, lat_grid, in_grid, cmap=cmap, vmin=0.0,
vmax=1.0)
       elif var == 'EA':
         cs = ax.pcolormesh(lon_grid, lat_grid, in_grid, cmap=cmap,
vmin=-0.3, vmax=0.3)
       elif var == 'UmDir':
         mag_grid =
numpy.array(input_vars_cases['Um']).reshape(nxgrid,nygrid)
         mag_grid = numpy.where(mag_grid=='*', numpy.nan, mag_grid)
         mag_grid = mag_grid.astype(numpy.float)
         uVec = numpy.array(input_vars_cases['UmX']).reshape(nxgrid,nygrid)
         vVec = numpy.array(input_vars_cases['UmY']).reshape(nxgrid,nygrid)
         step = 4
         lon_grid_q, lat_grid_q = lon_grid[::step, ::step], lat_grid[::step,
::step]
         uVec_q, vVec_q = uVec[::step, ::step], vVec[::step, ::step]
         Q = ax.quiver(lon_grid_q, lat_grid_q, uVec_q, vVec_q, angles='xy',
scale_units='xy', scale=1000)

       ax.set_axis_off()
       fig.savefig(out_path +var +'_overlay.png', transparent=False,
format='png')
    # end if in_grid all one value

    # make a colorbar for scalars
    if var == 'Hsig'  or var == 'UXO' or var == 'IB' or var == 'Um' :
      if numpy.nanmin(in_grid) != numpy.nanmax(in_grid):
         fig = pyplot.figure(figsize=(1.0, 4.0), facecolor=None,
frameon=False)
         ax = fig.add_axes([0.0, 0.05, 0.2, 0.9])
         cb = fig.colorbar(cs, cax=ax)
         cb.set_label(var, rotation=-90, color='w', labelpad=20)
         cbytick_obj = pyplot.getp(cb.ax.axes, 'yticklabels')
         pyplot.setp(cbytick_obj,color='w')
         # Change transparent to True if your colorbar is not on space :)
         fig.savefig(out_path+var+'_legend.png', transparent=False,
```

```
format='png')
         pyplot.close()

         if do_kmz_plots:
            make_kml(llcrnrlon=numpy.array(min(input_vars_cases['lon'])),
                llcrnrlat=numpy.array(min(input_vars_cases['lat'])),
                urcrnrlon=numpy.array(max(input_vars_cases['lon'])),
                urcrnrlat=numpy.array(max(input_vars_cases['lat'])),
                altitude=4000,
                figs=[out_path +var+'_overlay.png'],
colorbar=out_path+var+'_legend.png',
                kmzfile=out_path+var+'.kmz', name=var)

    elif var == 'UmDir' :
        if do_kmz_plots:
           make_kml(llcrnrlon=numpy.array(min(input_vars_cases['lon'])),
                llcrnrlat=numpy.array(min(input_vars_cases['lat'])),
                urcrnrlon=numpy.array(max(input_vars_cases['lon'])),
                urcrnrlat=numpy.array(max(input_vars_cases['lat'])),
                figs=[out_path +var+'_overlay.png'],
                kmzfile=out_path+var+'.kmz', name=var)


# Choose some different OUTPUT scenarios and plot
for key in output_vars_names.keys():
    nodes_in = list(nodes_info.keys()) # list of nodes in Netica object
    i_key = nodes_in.index(key)
    node_info.node = pynetica.NthNode_bn(all_nodes, i_key)
    node_info.name = pynetica.GetNodeName_bn(node_info.node)
    node_info.levels = pynetica.GetNodeLevels_bn(node_info.node)
    node_info.num_states = pynetica.GetNodeNumberStates_bn(node_info.node)
    # grab index of key in dict by converting keys to list
    key_ind = list(output_vars_names.keys()).index(key)
    cmap = colorbrewer.get_map(output_cmaps[key_ind], 'sequential', 5,
reverse=False).mpl_colormap
    for i in range(node_info.num_states):
        out_grid = output_vars_cases[key][:,i].reshape(nxgrid,nygrid)
        # Note:  5th TotalBurial (TB) bin is probability of 100% burial
        fig, ax =
gearth_fig(llcrnrlon=numpy.array(min(input_vars_cases['lon'])),

 llcrnrlat=numpy.array(min(input_vars_cases['lat'])),

 urcrnrlon=numpy.array(max(input_vars_cases['lon'])),

 urcrnrlat=numpy.array(max(input_vars_cases['lat'])),
                            pixels=pixels)
        cs = ax.pcolormesh(lon_grid, lat_grid, out_grid, cmap=cmap, vmin=0,
vmax=1)
        ax.plot(lon_grid[mx,my], lat_grid[mx,my], 'ko', linewidth=2,
```

```
markersize=8)
        ax.plot(lon_grid[mx,my], lat_grid[mx,my])
        ax.set_axis_off()
        fig.savefig(out_path
+key+str(node_info.levels[i])+'_'+str(node_info.levels[i+1])+'_overlay.png',
transparent=False, format='png')
        pyplot.close()

        fig = pyplot.figure(figsize=(1.0, 4.0), facecolor=None,
frameon=False)
        ax = fig.add_axes([0.0, 0.05, 0.2, 0.9])
        cb = fig.colorbar(cs, cax=ax)
        cb.set_label('Probability', rotation=-90, color='w', labelpad=20)
        cbytick_obj = pyplot.getp(cb.ax.axes, 'yticklabels')
        pyplot.setp(cbytick_obj,color='w')

 fig.savefig(out_path+key+str(node_info.levels[i])+'_'+str(node_info.levels[i
+1])+'_legend.png', transparent=False, format='png')
        pyplot.close()

        if do_kmz_plots:
          make_kml(llcrnrlon=numpy.array(min(input_vars_cases['lon'])),
             llcrnrlat=numpy.array(min(input_vars_cases['lat'])),
             urcrnrlon=numpy.array(max(input_vars_cases['lon'])),
            urcrnrlat=numpy.array(max(input_vars_cases['lat'])),
             figs=[out_path
+key+str(node_info.levels[i])+'_'+str(node_info.levels[i+1])+'_overlay.png'],

colorbar=out_path+key+str(node_info.levels[i])+'_'+str(node_info.levels[i+1])
+'_legend.png',
             kmzfile=out_path
+key+str(node_info.levels[i])+'_'+str(node_info.levels[i+1])+'.kmz',
name='Probability')

# make some plots for comparison at locatio of Calantoni Duck2015 data
TBProb = output_vars_cases[key].reshape(nxgrid,nygrid,node_info.num_states)
    ind = numpy.arange(node_info.num_states)
    fighist, axhis = pyplot.subplots()
    phis = axhis.bar(ind,TBProb[mx,my,:], width = 1, align = 'edge',
edgecolor = 'k', color = cmap._lut[-2], tick_label = node_info.levels[0:-1])
    axhis.set_ylabel('P')
    axhis.set_xlabel(output_vars_names[key])
    axhis.set_title('at DUCK2015 Quadpod')
    fighist.savefig(out_path+key+'.png', transparent=False, format='png')
    pyplot.close()
```