1 Steps

1. **Gen_Titan_Input_Samples_PHM_Montserrat_Take2.m**
2. **ExtractSampleOverRideFile.m** – run prior to titan to generate file titan reads modified input from, for montserrat take2 this calls the DEM making program
3. **run titan**
4. **down_sample_pileheightrecord.m**
5. **build_macro_emulator.m** – this could conceivably be run before or at the same time as steps 2 to 4 but if any simulations fail to complete, you will need to modify *uncertain_input_list.txt* to remove those entries and then rerun **build_macro_emulator.m**
6. **extract_mini_emulator_build_meta_data.m** – you tell it a simulation it pulls the neighbor information out of the macro emulator and writes it to a temporary file.
7. **build_mini_emulator.m** – you tell it the simulation number, it reads the temporary file generated in step 6. That tells it which *down_sampled_data.%06g* files to read. It builds that mini emulator.
8. **gen_random_macro_emulator_resample_inputs_Montserrat_Take2.m** – you tell it how many resamples you want if randomly generates them and writes them to *macro_resample.tmp* (you need to customize this for each hazard map)

   Step 8 and 9 can be run before or at the same time as steps 6 and 7.
9. **identify_mini_emulators_to_evaluate.m** – this reads the macro emulator and *macro _resamples.tmp* file and finds the simplices containg each resample and the barycentric coordinates, sorts the resamples by the simpleces containing them, and writes this into *macro_resample_assemble.inputs*.
10. **make_initial_hazard_map_Montserrat_Take2.m** – generates the zero'd phm file. This can be run at **any** time up to this point (i.e. it can be the first matlab file you run). It contains the numerical value at the hazard criteria i.e. **0.2** if the criteria is **pileheight** $>$ **0.2m**.
11. **extract_macro_simplex_resample_inputs.m** – you tell it:
    - a **simplex** (not sample) number;
    - the name of any compatible hazard map file (can be created by step 10 but does not still have to be zero'd);
    - how many (max) resample points you want per file, this last one lets you split work between multiple processors/nodes when there is a large number of resamples/simplex, it creates "unique" random keys for each file.

12. **evaluate_mini_emulator_mean.m** OR **evaluate_mini_emulator_mean_and_variance.m** – for each file generated in step 11 (minimum of 1 per simplex) you will need to evaluate the mini-emulators located at the nodes of that simplex. Note that different simplices split the resamples so "in general" you will need to evaluate each mini-emulator a number of times greater than or equal to the number of simplices it's a node of but the re-sample data is grouped by simplices not mini-emulators. Exception when the dimensionality is "high" some simplices may contain no resample points, in this case **extract_macro_simplex_resample_inputs.m** would have returned an empty matrix/string at the command line and NO output files would be written. The idea is to read *macro_resample_assemble.input* one line at a time, compare the current simplex to the previous simplex, if they are different evaluate **extract_macro_simplex_resample_inputs.m** and the mini emulators at the nodes of the new simplex. If they are the same then don't evaluate that simplex because you are already done with it.

13. **assemble_minis_to_macro_to_phm.m** – for each file produced in step 11 (see step 12 for more explaination) evaluate this .m file once, passing it:

    - the name of one of the mini-emulator outputs (step 12) resulting from the evaluation of that file (produced in step 11);
    - the name of any compatible phm file (it can be created in step 10 but it does not still need to be zero'd)

    This function will automatically read in all the other matching micro emulator outputs produced in step 12, combine them into a macro-emulator output, evaluate the hazard criteria based on the macro emulator output, and generate a "component PHM" that corresponds to the file generated in step 11. Note that it does not write the macro-emulator output to file because it would be very time-consuming/write-intensive and we don't need to keep that information. The component PHM is very small by comparison. The function needs to be slightly modified (comment/uncomment a few lines) when switching between **evaluate_mini_emulator_mean.m** and **evaluate_mini_emulator_mean_and_variance.m** in step 12.

14. **merge_probability_of_hazard_maps.m** – first "real" argument is the name of a "permanent" hazard map, the second "real" argument is the name of a "temporary" hazard map. It reads in the permanent and temporary hazard maps adds the contents of the "temporary" phm to the "permanent" phm and outputs the updated "permanent" phm. Note that "permanent" is used loosely, it means cumulative. You could add them up in a serial loop or a tree fashion or some combination of trees + serial loops.

15. **view_phm.m** this is a raw/primitive code that reads in a phm, calculated the spatially varying probability and non-confidence from the sums contained in the phm, and does a rudimentary plotting of the probability non-confidence, this file should be modified to produce pictures with an appearance that is suitable for the application/ purpose for the map.

2 Function description

1. **Gen_Titan_Input_Samples_PHM_Montserrat_Take2.m** –this function needs to be modified for each PHM produced. It generates the inputs for the Titan simulations from which the hierarchical emulator will be constructed. The essential idea is to "uniformly" cover the RANGE (not distribution) of the simulation inputs so that there will always be simulations/samples nearby any possible re-sample point. Since emulators are generally poor at EXtrapolating, it is a good idea to slightly enlarge the sample range. This helps to ensure that "all" re-samples will lie within the convex hull of the sample points. We use a space-filling latin hypercube sample design (generated by BinOptLHSRand.m) which is mapped to the range of inputs. Some tips:
   - sample should be uniform in log(volume) rather than volume
   - if you want to generate a starting location that is uniformly distributed on a circle you should map one dimension to angular direction, and NOT linearly map the second to radius R because that will not be uniform. Instead you should linearly map the area to the left of the unmapped point to the area of a circle with radius R.
2. **down_sample_pileheightrecord.m**
3. **build_macro_emulator.m** – the macro emulator is simply a Delaunay tesselation of the simulation /sample inputs. This tesselation is used to define the support of each mini-emulator and to recombine the mini emulator output into macro emulator output. There will be 1 mini emulator centered about each simulation/sample. The support of each mini-emulator are only those samples that can be reached within 1 or 2 hops (choose 1 or 2 hops, 1 hop is cheaper) along edges of simplices in the tesselation. To save execution time later, the neighborhood/support of each simulation/ sample is computed for ALL simulations/samples in this function and written to the macro emulator file. If starting location is/are 2 of your uncertain variables it is a good idea to have them in (UTME, UTMN) form rather than radius and angular direction because delaunayn.m doesn't do periodic tessellation so you get a "boundary" at 0°/360° that may have gaps (convex hull of data points and don't have samples exact at corners of cube) and WILL get some high aspect ratio simplices there.
4. **extract_mini_emulator_build_meta_data.m** – this function reads the macro emulator and extracts the neighborhood information for 1 specified simulation/sample/mini-emulator. It also calculates scale-factors that are used to non-dimensionalize the input dimensions independently. "Bounds" on acceptable correlation lenghts for mini-emulator (macro dimensions only) are computed in terms of the nondimensional radius of a hypersphere with the same "content" (hypervolume) as the 1 hop neighborhood of the specified simulation/sample/neighborhood. This **radius** is always calculated from 1 hop macro neighborhood even if more then a 1 hop neighborhood is used to construct the mini-emulator. Non-uniformity in sample spacing tends to "corrupt" the estimation of correlation length bounds and you want a local (nonstationary) answer. Not so important

with uniformly spaced samples but it is important if you ever want to do adaptive sampling/simulation.

5. **build_mini_emulator.m** – reads in the build mini emulator meta data from the specified simulation/sample /mini-emulator, then reads in the down sampled data files for each of the simulations in the 1 or 2 hops neighborhood defined in the macro emulator and extracted into the build mini emulator meta data. Even if the macro neighborhood is 1 hop, I use 2 hops to define the tensor product of uncertain inputs and physical space (i.e. that would be 2 spatial hops in the central simulation, 1 spatial hop in the simulations that are 1 stochastic (macro) hop away from the central simulation, and if you use 2 macro hops then there would be zero spatial hops from simulations 2 macro hops away i.e. only the spatially nearest covering node from simulations 2 macro hops away) but that's getting a little ahead of this explanation...the first thing is that as each downsampled data file is read in, the macro component of the tensor product neighborhoods for each micro-emulator (which are centered around covering nodes in the central simulation's downsampled data file) are constructed, a "macro hop" between simulations takes you to the spatially nearest covering node in the destination simulation/downsampled data. The nearest **covering** node is important for assembly of mini-emulator output into macro-emulator output at the boundary of the flow, i.e. flow outlines, i.e. it "interpolates" better when a resample is outside the boundary of one simulation's flow outline but inside the boundary of another simulation/mini-emulator-node of a macro simplex. It also guarantees that you won't be missing data you need to construct micro-emulators (too many zero outputs makes for bad micro-emulators, not enough data to "interpolate" well). Micro-emulator construction: generate a few guesses for correlation lengths within the bounds (for macro inputs these were set in **extract_mini_emulator_build_meta_data.m**, for spatial dimensions these are calculated in this, **bulid_mini_emulator.m** file). If the resulting correlation matrix has too large a condition number the take a few steps in the direction of steepest descent of condition number to make it smaller, within those constraints choose correlation lengths that minimize $\hat{\sigma} - fast$ fast. Minimization is done using Newton's method. Choose the best answer from all my few guesses. Storing inverses of correlation matrices would make the mini emulator files too big so only store $\underline{\underline{R}}^{-1} \cdot \underline{\epsilon}$ this what you need if you only want to evaluate the mean. If you later also want to evaluate the adjusted variance it will be recomputed then. Oh and during the step to decrease condition numbers *increase_rcondR_to_minrcondR()* and its subfunction *eigen_rts_dist()* only use the first 2 derivatives of eigenvalues.The loop nesting in *eigen_rts_dist()* gets very expensive very fast. It's less expensive to take more smaller steps to decrease the condition number.

6. **gen_random_macro_emulator_resample_inputs_Montserrat_Take2.m** – this m file generates the re-samples inputs at which the hierarchical emulator will be evaluated to compute the probability of hazard map. It is similar to **Gen_Titan_Input_Samples_PHM_Montserrat_Take2.m**. It calls BinOptLHSRand.m to generate a spacefilling latin hypercube design

with a large number of re-sample points. We typically use 2^{17} or 2^{18} re-samples but you may want to increase this particularly if you increase the number of macro-inputs beyond 4. Why? because for 4 macro inputs and 2^{18} points, some macro-simplices ended up containing 1 or probably/ almost certainly no re-sample points, so cost to evaluate none would be small, most of the "work" is input-output.

Important. These re-samples must be drawn from the DISTRIBUTION (not just range) of the uncertain inputs. To get better coverage of the volume input dimensions we use importance sampling. You need to edit this for each PHM/ distribution change.

7. **mytsearchn.m** and **identify_mini_emulators_to_evaluate.m** – this reads in the re-sample points and identifies the macro-simplices containing them and computes barycentric coordinates (which will be used to assemble mini- emulator output into macro-emulator output) and sorts the re-samples by the macro-simplices that contain them (which makes extracting the macro simplex resample inputs very fast, you just move sequentially along the list instead of having to search the whole lost for each macro-simplex).

   The locating of the simplices containing the points is done by a function I wrote mytsearchn.m to replace the default Matlab "tsearchn.m". I developed the macro emulator using an old version of Matlab that was then installed on omega.eng.buffalo.edu and that old default algorithm was very slow (weeks for a few thousand re-samples). My algorithm for **mytsearchn.m** uses a multi-dimensional binning quick sort to introduce locality/sequentiality into both the sample and resample points. I travel along the sequence of sorted re-sample points while marching along the sequence of sample points and that gives me a very good first guess at which macro- simplex contains the next re-sample point (probably the same as the last one which lets us reuse work already done). If the previous macro simplex doesn't contain the next re-sample (which you can check by computing the bary centric coordinates for that simplex and re-sample, if all of the bary centric coordinates are positive then the resample is in the simplex) then you look for neighbors of the previous simplex in particular you travel in the direction of the most positive nodes/barycentric coordinates. If you get to a boundary you can tell this by all but one node having positive barycentric coordinates and no other simplex sharing all of the positive nodes. This works because the Delaunay tesselation provides a convex hull (there are no holes in the tesselation). It can break down when the resample is outside the convex hull and is beyound the "corner" of the hypercube tesselation in multiple dimension. i.e. there would be more than one node with barycentric coordinates). That doesn't keep it from working, it just slows the search down becasue it has to check "all" the neighboring simplices but it can tell when its getting closer or farther away and knows enough not to recheck simplices that has already checked for this resample because I keep a list. To reitterate the reason that my mytsearchn.m algorithm is so fast is that it has very good initial simplex guesses for each resample point and it has very good guess for which neighbooring simplex to check next and it knows enough to stop looking i.e. it can figure out when the re-sample is outside the convex

hull of samples. Matlab has since improved their tsearchn.m algorithm so that it runs in the same order of magnitude of time as mine, but mine is about 20% faster when is not compiled/ compiler optimized and theirs is compiled and compiler optimized.

8. **make_initial_hazard_map_Montserrat_Take2.m** – this is very straight forward, it creates a zero'd PHM file using the corners of the map you want to make and a specified number of cells for each direction. These generate the spatial points at which you will evaluate the ensemble emulator. The PHM contains the numeric value of the hazard criteria e.g. 0.2 for 0.2m but it could conceivably be modified to contain a formula for a hazard criteria, which would be especially useful for automating the entire procedure and muti-hazard analysis (the outputs of multiple ensemble emulators or an ensemble multioutput emulator) this could also be modified to indicate which mini emulator evaluation function should be used although it may be easier to do that with a default wrapper function that calls whatever mini-emulator evaluation function you want.

9. **extract_macro_simplex_resample_inputs.m** – given a specified macro simplex (not sample) number it pulls that information the re-sample points for that macro simplex out of the macro-resample-assemble.inputs file and pulls the east north coordinates out of the PHM file (also specified) and writes that information to one or more data files name *eval_resamples_for_simplex_%d.%8f* (where %d is simplex number and %8f is randomly generated unique identifier) how many files it splits the macro resamples among depends on a user specified maximum number of re-samples per file. This is more important when you have a larger number of resamples and a small number of macro-dimensions (and therefor macro simplices) it alows you to split a large number of resamples in a simplex to multiple files that can be evaluated concurrently on different nodes of a cluster. Be careful if you try making a PHM for table top experiments east or north map coordinates currently have centimeter scale resolution %.2f.

10. **assemble_minis_to_macro_to_phm.m** – this is pretty straight forward:

    - read in the mini-emulator evaluations
    - make sure they match (that's what the random unique identifier is for see extract_macro_simplex_resample _inputs.m)
    - combine them based on their barycentric coordinates within the macro simplex
    - evaluate the hazard criteria – Indicator function can be binary or fuzzy/probability you can evaluate a probability Indicator function using the error function (for normal distributions that is). You need to comment/uncomment a few lines to switch between binary and probability indiactor functions and also need to use the matching mini emulator evaluator function mean and variance will work for either but takes longer than just mean.
    - add the Indicator's first and second moment to see the sums of first and second moment of probabilities

- write first and second moment probability sums to the component PHM file. This is much faster than **writing** macro emulator evaluations which you don't need "permanently" any way.

11. **merge_probability_of_hazard_maps.m** – straight forward:

    - read 2 "component" PHMs - make sure they don't have any of the same macro-resample points (don't want to double count)
    - **add** their sums of first and second moments of probabilities (actually it's a sum of indicator function evaluations rather than probabilities)
    - write there sums of first and second moment indicator function evaluation to amerged PHM file.

12. **view_phm.m**

    - read in the phm which is essentialy just the sums of the first and second momnets of indicator function evaluations for each point on the map.
    - use the first and second moment sums to calculate mean probability and second CENTRAL moment of probability for each east north point on map
    - "non confidence" is $\frac{\sigma_p}{p}$
    - plot probability and non confidence. Makes whatever pretty picture you want. What I've got written in there will show you how to convert a numerical value into a color value that you can plot with patch or whatever image can be useful when you want to display a satelite image for terrain.