Coupling Oasis LMF with CLIMADA

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Coupling Oasis LMF with CLIMADA

Term Paper

Robert Blass

March 31, 2021

Advisors: Prof. Dr. D. N. Bresch, Dr. C. M. Kropf
Department of Environmental Systems Science, ETH Zürich

Abstract

Modelling the risk of weather and climate scenarios is important when estimating the socio-economic impact and the risk arising from extreme weather situations. Frameworks performing such models often specialize in a sub-field of weather risk modelling. The coupling of frameworks can therefore offer new opportunities. In this term paper, the goal is to enable interoperability between the two frameworks CLIMADA and Oasis Loss Modelling Framework (LMF). This is achieved by calling and executing models with the Oasis LMF computational kernel from CLIMADA. Resulting from this coupling, data translation from CLIMADA to Oasis LMF is now possible. Further, it allows an alternative uncertainty treatment in CLIMADA. Concluding, the achieved interoperability offers both frameworks new possibilities for estimating risks of weather and climate.

Contents

Chapter 1

Introduction

1.1 Climate Risk Modelling Software

Assessing the risk of extreme weather situations is important to gain understanding of the impact catastrophes can have and how probable extreme weather situations are. It should therefore help in the decision process when selecting measures to reduce the risk. Another usage of risk assessment is to better understand the risk in insurance and selecting the fitting offer for different regions.

In this work the focus lies on two softwares helping to estimate the impact for affected areas and quantifying the risk.

CLIMADA Here a definition of CLIMADA is recalled [1]. CLIMate ADAption (CLIMADA) is an open-source platform for probabilistic risk modelling and options appraisal. It integrates hazard, exposure and vulnerability to assess risk and quantify the socio-economic impact. Supporting multi-hazard calculations and providing state-of-the-art probabilistic modelling, it allows to estimate the incremental increase in impact and risk coming from economic development and climate change.

Oasis Loss Modelling Framework Oasis Loss Modelling Framework (LMF) is an open-source catastrophe modelling framework and its development is mainly focused on the (re-)insurance sector [4]. It uses the modelling engine Oasis LMF kernel tools (*ktools*) and models the risk of multiple hazards and vulnerabilities [4]. This kernel performs the core calculations of the framework which include damageability distributions and Monte-Carlo sampling of ground up loss.

1.2 Interoperability of Frameworks

Interoperability defines "the degree to which two products or programs can work together or the quality of being able to be used together" [5]. This term paper describes a first step in interoperability of the introduced software programs. The fundamentally different setups of these programs prove challenging due to, for example, the coding languages, data handling, data storing and risk assessment metrics.

1.3 Advantages of Coupling Systems

Coupling two frameworks allows each framework to take advantage of the implementation of the other. Since both frameworks stem form the insurance background but have been developed into different directions, they can work in a complementary fashion.

On the one hand, this term paper should enable the translation of hazard data from CLIMADA to Oasis LMF, resulting in more open data availability for Oasis LMF users. On the other hand, the implementation should allow CLIMADA users to take advantage of the Oasis LMF *ktools* engine. This enables an alternative uncertainty treatment in risk modelling and performing calculations.

1.4 Implementation of Interaction

Since the implementation should allow interaction with the Oasis LMF kernel from CLIMADA, the coupling of the frameworks needs to be implemented in the latter. The corresponding part of CLIMADA to *ktools* is the method *impact.calc* in the package engine. The object *Impact* of this package which includes this method will therefore be expanded by the new method *impact.calc_ktools*. This method should take care of translating the data to ktools, executing computations with *ktools*, and translating the output back into CLIMADA format.

Chapter 2

Overcoming Differences in Modelling Techniques

First in this chapter the concept and characteristics of each framework are described individually. The focus lies on the properties that are important to achieve the mentioned goals in Section 1.3 and to highlight the differences of the frameworks. Afterwards, the data translation from CLIMADA to Oasis LMF, performing computations with Oasis LMF and transferring the results back are explained.

2.1 CLIMADA - Short Overview

This section is based on the work [1]. The fully probabilistic risk assessment model CLIMADA is based on the following definition of risk:

$$risk = probability \times severity,$$

where severity can be approximated as follows:

$$severity = exposure * f_{imp}(hazard\ intensity),$$

where f_{imp} is the impact function which characterizes the damage of a hazard on an exposure. The implementation of CLIMADA in Python contains three main packages: *hazard, entity* and *engine*.

hazard In CLIMADA, the *hazard* package includes the implementation of different hazard events, for example tropical cyclones, floods or droughts. For this project the focus lies on tropical cyclones. A statistical understanding of the hazard occurrences is given which allows to define a frequency of the events. The data for simulating hazards is either taken form historical data or by generating a probabilistic event set.

entity Socio-economic aspects of a risk assessment in CLIMADA are covered by the package *entity*. It includes an object exposures which quantifies assets with a value for given coordinates.
Another important part of this package are the impact functions. Defined for different exposures and hazard types, they approximate the relative impact from hazard intensities on exposures, such as shown in Figure 2.1. This is done in the method *calc_mdr* by coupling the mean damage ratio (MDR) of exposures to hazard intensities. The impact function is often continuous or returns the linearly interpolated values when it is only defined for point values. For example in Figure 2.1, for a wind intensity of $60m/s$, 26% of the exposed value is destroyed.

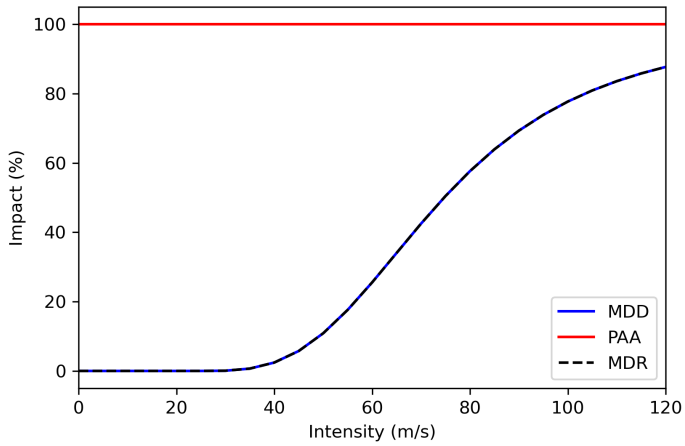

Figure 2.1: Example of an impact function. MDD stands for mean damage degree, PAA for percentage of affected assets and MDR for mean damage ratio.

engine The object *Impact* which is the main part of the *engine* package, computes the impact of a hazard (set of historical or simulated (probabilistic) natural hazard events) on exposures. The impact function of the package *entity* is used to calculate the relative impact for a hazard intensity on exposures. This multiplied with the exposure value results in the impact:

$$x_{ij} = val_j * f_{imp}(h_{ij}|\gamma_j),$$

where x_{ij} is the impact due to event i at location j, val_j the value of exposure at location j, f_{imp} the impact function which takes as input a hazard intensity h_{ij} given γ_j, namely characteristics of the exposures.

2.2 Introduction to Oasis LMF

Following from the set objectives in Section 1.4, the new method *impact.calc_ktools* should be able to perform the same computations as the method *impact.calc*. Those computations are performed in Oasis LMF by the kernel toolkit *ktools*. Therefore, the focus lies on *ktools* only and will be introduced below. This is done with the help of a *ktools* presentation [2] and its documentation [3] which are cited multiple times in this section.

2.2.1 Name of Variables

To help the reader's orientation the variable names of CLIMADA and *ktools* are compared in Table 2.1.

CLIMADA	*ktools*
impact	loss
event	event
exposure	coverage
exposure value	total insured value
centroid	area peril
impact function	vulnerability function

Table 2.1: Comparison of variable names.

2.2.2 Implementation

ktools is implemented in C++ and it defines the processing architecture and data structures. "The kernel is provided as a toolkit of components which can be invoked at the command line. Each component is a separately compiled executable with a data stream of inputs and/or outputs. The principle is to stream saved data files through calculations in memory" [3].

2.2.3 Data Structure

The implementation approach used in *ktools* means for the data handling that input data has to be subdivided into multiple files. When invoking specific executables, its necessary data files need to be previously stored in defined data folders.

ktools saves hazard properties in discrete bins, ranges of values grouped together and recognizable by a bin number. The vulnerability function connects these bins with the probability they occur with. In *ktools* exposures are saved by connecting its exposure id with the exposed value. Therefore, some files mainly focus on connecting the correct information from multiple

files while others include the actual data needed for computations. The important files to gain interoperability are explained here and for every file there is an example in the Appendix A.1.

damage bin dictionary This dictionary defines how the effective damageability is discretized on a relative damage scale. As can be seen in Figure 2.2 on the right, the damage bins are defined to be $0-40\%$ and $40-100\%$. This means for the latter bin that the damage lies between 40 and 100 percent. The number and sizes of bins can be defined by the user. In the file the bin index, start (bin_from), end (bin_end) and interpolation of every damage bin need to be defined. The quantity interval_type is irrelevant.

footprint This file introduces the concept of intensity bins. The user selects the number of intensity bins and how they are connected to the damage bins. Since this connection is given by the user, there is no need for defining specific intensity bins. These bins can be seen in Figure 2.2. The implementation allows multiple intensity values per event and area perils. Further, one has to insert the probability p^f that an event occurs with an intensity in the corresponding intensity bin. Since this probability needs to be specified for every combination separately, it cannot be inserted as a distribution function. To clarify, this probability is not shown in Figure 2.2.

vulnerability The vulnerability file can include multiple vulnerability functions. An example of a vulnerability function can be seen in Figure 2.2. The probability values shown there have to be defined for every vulnerability function, intensity bin and damage bin. It specifies the probability p^v of an intensity causing a certain damage. For example in Figure 2.2 the intensity $65m/s$ causes a damage of $0-40\%$ with a probability 0.8. Another vulnerability function in the same model would have the same intensity and damage bins but the probability distribution of intensities causing certain damages would be different.

coverages For every coverage a coverage id and its total insured value need to be specified.

events Only the event ids are listed.

items In this file a list of exposure items can be given for which the loss should be calculated by *ktools*. An item is specified by a a coverage, area peril and vulnerability. Further, it can be put into correlation groups which is a collection of items defined by the user. When items are in the same group the sampled damage is fully correlated and otherwise fully independent (sampling damages explained in Subsection 2.2.5).

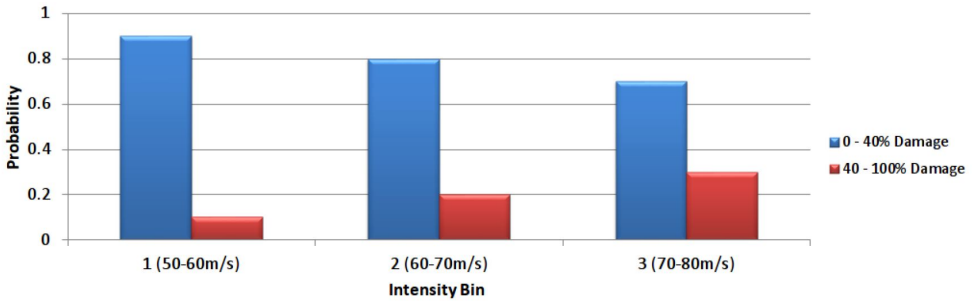

Figure 2.2: Example of a *ktools* vulnerability function. Specified intensity bins and with what probability which damage range they cause. (Ben Hayes, Presentation "Oasis under the Hood", slide 8, [2])

gul summary xref Here the summing of coverage losses are defined. There can be up to ten summary sets defined where for every coverage id and summary id a summery set id needs to be specified. The summary set id specifies the summary level of grouping, for example grouping at site level. If there is for every coverage id a separate summary id the summation has no impact on further computations.

occurrence The occurrence file assigns to every event a date and a period number it occurs in. The period number can be specified and the occurrence can be represented as an offset number of days to a base date.

These files are all stored as *csv* files in defined folders. To run computations with *ktools* they need to be converted to binary files which can be done with the help of separate *ktools* executables.

2.2.4 Sequence of Computations

As all necessary files to run *ktools* are converted to binary files, the core component executables of the engine can be invoked. These executables perform different computations in a pipeline manner, which means that an executable's output (and some additional files) are the input of the following one. So they build a queue and hand the results to the next waiting via storage. After every part of the queue the data is saved in storage. An overview of the most important executables:

eve This executable is the beginning of every pipeline and reads in the events binary. It saves the list of event ids.

getmodel It combines the probabilities p^f and p^v defined in the files *footprint* and *vulnerability*. How this probabilities are combined is explained in Detail in Subsection 2.2.5.

gulcalc Here the loss of models are computed. There is a deterministic and a sampling approach to perform this computation. How this computations are performed is explained in more detail below in the Subsection 2.2.5.

summarycalc The main task of this executable is to sum information together. The degree of summation can be selected by choosing the summary id and summary set id in the *gul summary xref* file.

eltcalc This executable calculates the sample mean and standard deviation of the loss for every event id and summary id. As the loss is already computed in the executable *gulcalc*, *summarycalc* and *eltcalc* only perform summations of losses and no actual loss computations.

2.2.5 Loss Calculation

The computation of loss is the most important part in a kernel like *ktools*. For this computation, the probabilities (here enhanced with subscript i and d to show that the uncertainty lies in intensity, respectively damage for each probability) defined in the *footprint* and *vulnerability* files play together:

$$p_{id} = p_i^f * p_d^v,$$

where p_{id} is the probability of an intensity i occurring and causing a certain damage d, p_i^f the probability mass for the intensity bin and p_d^v for the damage bin, respectively. That computation of p_{id} is done for every combination of event, area peril and vulnerability function. p_{id} is then summed over the intensity bins:

$$p_d = \sum_i p_{id},$$

which results in p_d, the probability of a damage bin d for every event, area peril and vulnerability combination.

Since the probability distribution is discrete, the values can be summed up to build a cummulative distribution functions (CDF), which is done in the *getmodel* executable. They are summed over intensity and damage bins which results in a CDF for every event, area peril and vulnerability.

As explained, two approaches are possible in the *gulcalc* executable. Both computations are applied independently of each other. In the deterministic integration approach the calculated damageability probability distribution is integrated, yielding the so called damage factor. That factor multiplied with the total insured value results in the loss:

$$x_{\varepsilon\gamma} = tiv_\gamma * d_{\varepsilon\gamma}, \tag{2.1}$$

where $x_{\varepsilon\gamma}$ is the loss at a given event ε and coverage γ, tiv_γ is the total insured value and $d_{\varepsilon\gamma}$ is the damage factor. This approach does not return a standard deviation since integration does not allow for uncertainty computations.

Choosing the sampling instead of the integration approach does not change anything up to and including calculating the CDF's. Then, to sample one of the calculated CDF's a random number generator selects a number between zero and one, as done in Figure 2.3 with number 0.82. This number in the CDF selects the damage bin it lies in, as seen in the upper part of Figure 2.3, the damage bin 2. The selected number lies in the chosen damage bin with a specific ratio (31% in Figure 2.3 bottom left). Applying this ratio to the selected damage bin results in the damage factor (0.584615 in Figure 2.3). Through this, a single value of damage is chosen from a range of damages when sampling. As in equation 2.1, multiplying this factor with the total insured value yields the loss for this sample.

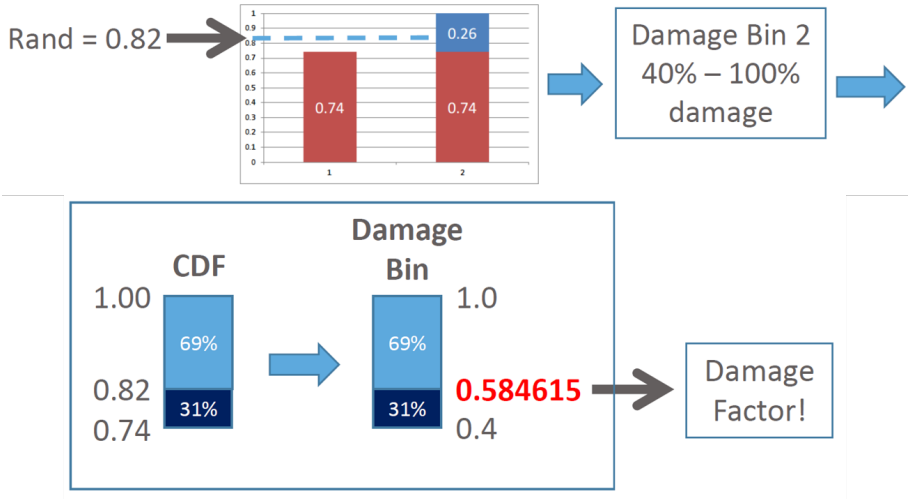

Figure 2.3: Graphical illustration of sampling approach when calculating loss. Select a random number, this samples the CDF, which then yields the selection of damage bin. Afterwards through applying the same ratio to the damage bin, the damage factor of that sample is found. (Ben Hayes, Presentation "Oasis under the Hood", slide 21, [2])

From this approach a loss results for every sample. From all these samples the statistics mean and standard deviation can be computed. How many samples should be used can be selected in the *gulcalc* executable.

Results

An example of the output of *ktools* (output of executables *eltcalc*) is shown in Table 2.2. The summary id represents the summary levels chosen by the user in *gul summary xref*. Type "1" indicates that the deterministic loss

computations from Subsection 2.2.5 are performed. With "2" that the mean and standard deviation are computed via sampling. Further, the exposure values of the summary ids is given. If in the *gul summary xref* file for every coverage a separate summary id is chosen, the summary id corresponds to the coverage id. This will be used in the next Section 2.3.

summary_id	type	event_id	mean	standard_dev	exposure_value
1	1	1	20'500	0	30'000
1	2	1	18'393	1'536	30'000
2	1	1	8'909	0	10'000
2	2	1	12'106	2'503	10'000
1	1	2	43'500	0	50'000
1	2	2	38'562	5'236	50'000
2	1	2	43'500	0	90'000
2	2	2	96'912	6'683	90'000

Table 2.2: Example of an *eltcalc* output. There are two summary levels defined. Type indicates deterministic "1" or sampling "2" approach. Then, for every such combination with all events the computations are performed. The standard deviation is zero for deterministic approach and last column shows the exposed value for each combination.

2.3 Translation from CLIMADA to ktools

In this Section the implementation of the data translation from CLIMADA to *ktools* is explained. To check that the translation worked, it is necessary to perform the loss computations with *ktools* and compare them with the CLIMADA results. Thus, also calling the kernel and reading the solution back to CLIMADA is explained here. The uncertainty quantification is not part of this Section.

2.3.1 Calling Executables from CLIMADA

First of all, *ktools* needs to be installed locally to be able to communicate with it. Afterwards, calling executables is achieved by using the Python module *subprocess* in CLIMADA, which invokes *ktools* executables from the command line. Using that approach, the standard user does not need to work with the command line.

This implementation is done in the method *compute_impact_ktools* and differs between Windows and Unix based operating systems, since Windows cannot cope with pipelines. Further, the path to the executables differ in the operating systems. In Unix based systems the path is found automatically, in Windows the implementation requires user input.

2.3.2 Data Handling

The next step is to adapt the data system of CLIMADA where information is stored in objects, to the storage based system of *ktools*. For every file that *ktools* requires, there is a separate method implemented which translates data.

In every method, data is filled into a *csv* file which is saved in storage and then transformed to a binary file. Therefore, at the end of every method the method *data_conversion_ktools* is called which does exactly that. Since this method invokes executables that transform the *csv*'s to binary files, it is implemented in a similar fashion as *compute_impact_ktools*.

2.3.3 Data Translation

Here it is explained how to translate the CLIMADA data to fill the *csv* files needed by *ktools*. Therefore, the different concepts for exposures, impact functions and hazards are matched here.

Translate Exposures

As the concept of exposures and coverages is similar in both frameworks, the translation is trivial. It is mainly implemented in the method *generate_coverages*. The coverage id works as an index to combine the total insured value from the file *coverages* with the area perils from the file *items*. This combination is needed for the loss computation. As mentioned in Subsection 2.2.5 Paragraph Results, if for every coverage id an new summary id in the file *gul summary xref* is defined, the coverages are not summed together. Than, *ktools*' output is similar to CLIMADA's output which is necessary to compute the same result metrics for both frameworks.

Translate the Impact Function

The biggest difference between CLIMADA and *ktools* is the impact function. Whereas in CLIMADA the impact function computes an impact percentage for every intensity, *ktools* has for every intensity multiple damages it could cause. Also the intensities of a hazard are discretized into bins in *ktools*. These are two fundamentally different concepts and need for a translation, specifically a discretization of the properties in CLIMADA.

The discretization of the impact function is implemented in *generate_damage_bins* and is shown in Figure 2.4. The blue line shows the impact function from CLIMADA and the red line the constructed intensity bins for *ktools*. The latter is a step function and takes for example the intensities $50 - 60 m/s$ as intensity bin 6, which is marked blue. Then, for every such intensity bin the damage is calculated with the help of the CLIMADA method *calc_mdr*.

This impact percentage for the blue bin is 18% as can be seen in Figure 2.4 and 2.5. Figure 2.5 shows the three colored bins of the CLIMADA impact function in the *ktools* representation of an impact function.

Through this implementation, in *ktools* is only one damage bin possible for every intensity bin. This in normal in CLIMADA thinking, but a restriction in *ktools* where all combinations could be possible as seen in Figure 2.2. This restriction is applied in the method *generate_vulnerability* by defining only the intensity and damage bins when they have the same number.

Because there is only one combination of intensity and damage bin possible, the probability p^v that an intensity causes the corresponding damage has to be one. This probability is shown for three example intensity bins in Figure 2.5 and is inserted in the file *generate_vulnerability*.

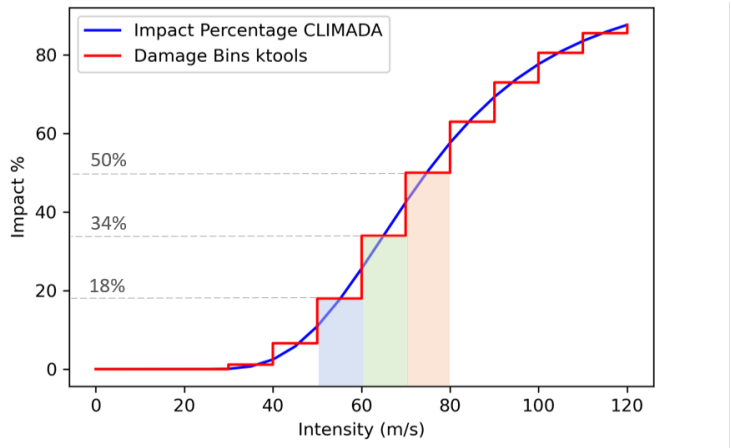

Figure 2.4: CLIMADA's impact function and *ktools* discretized version. The intensity is put into bins, 3 colored examples shown.

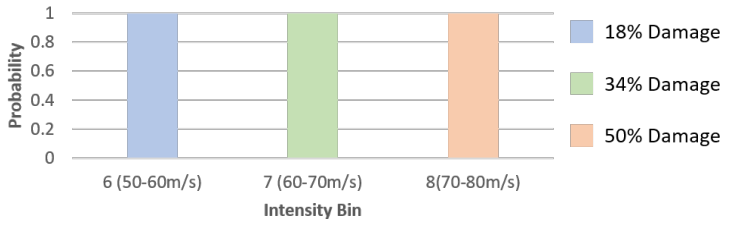

Figure 2.5: Three examples of intensity and damage bins corresponding to colored examples of Figure 2.4. When translating CLIMADA data (with no uncertainty) to the *ktools* impact function, the probability of the intensity causing the specific damage is always one.

A direct translation of the CLIMADA impact function to a multi-damage bins function in *ktools* is not readily achievable because in CLIMADA one hazard intensity always results in the same damage. Going from one damage to multiple damage bins is not defined and therefore cannot be done. However, one can instead directly define a *ktools*-impact function which is then translated to *ktools*. This allows to perform an uncertainty computation, as will be seen in Section 3.2.

Translate Hazards

Assuming the translation of the impact function from CLIMADA to *ktools* has worked in the above explained manner, the translation of the hazard data is only filling the values in the generated bins. This is mainly implemented in the file *generate_footprint*. This file also requires to fill the probability p^f which is not the same probability as in Figure 2.5. Since standard CLIMADA models do not account for this uncertainty it should be filled with one, when translating data.

2.3.4 Transfer Results

Computing the same metrics for both frameworks makes the results comparable and allows to assess the quality of interoperability. The metrics that can be computed after *ktools* modelling are the impact at event (*at_event*) and the expected annual impact (*eai_exp*) [1]. Those metrics can only be assessed with limitations as explained in Section 2.3.5.

To calculate the metrics of the ktools models, its results need to be imported into the CLIMADA structure. The method *exp_impact_ktools* does that. The loss described in Section 2.2.4 (Paragraph *eltcalc*) is given per event and area peril. Therefore, summing over the exposures results in the loss per event and summing over the events results in the expected annual impact. This is also computed in the method *exp_impact_ktools*.

2.3.5 Unsolved Translations

Unsolved translations remain when a property of the one framework has no correspondence in the other. The CLIMADA's hazard property fraction of affected assets is not defined in *ktools* and is therefore not translated to *ktools*. The fraction of affected assets would have an influence on the metrics *at_event* and *eai_exp*. A possible solution is to restrict oneself to only hazards with the neutral fraction one. Another possibility is to multiply the computed loss with the corresponding fraction but this has not been achieved thus far.

Since *ktools* does not have a statistical representation of hazards, the frequency of a CLIMADA hazard has no matching attribute in *ktools*. Thus,

the frequency cannot be integrated into *ktools* but it can be multiplied afterwards with the *ktools* result for the correct computation of *eai_exp*. Although the results correspond upon multiplication, the drawback is that the quantity frequency is not directly translated and thus cannot be used in direct conjunction with the hazard and vulnerability uncertainty calculation of *ktools*.

Another problem that occurs during translation are the different ids used in *ktools*. For example, the group id in the *items* file has no found impact in the calculations. Similarly, in the file *gul summary xref* the summary set id does not have an impact on the computations of the translated data. These are only observations and implemented in a manner to perform correct translations, but those indices are not yet fully understood.

Chapter 3

Results

The two advantages that arise from interoperability are clearly distinguished in this Section and their results can be analyzed separately. Firstly, the results for translating data are investigated. Secondly, the uncertainty computation is shown.

3.1 Comparison of Results after Translation

With the translation methods shown in Section 2.3 it is possible to get the same metrics *at_event* and *eai_exp* up to a discretization error with both frameworks after data translation. This error comes from the discretization of the CLIMADA impact function explained in Subsection 2.3.3. The following example assumes that the unsolved translations in Subsection 2.3.5 explained, are excluded from the modelling data. If data includes these unsolved translations, the metrics *at_event* and *eai_exp* cannot be reproduced with *ktools*.

3.1.1 Example to Analyze Accuracy

This example is to analyze the accuracy of the computed metric *at_event* after the data translation to *ktools*, loss computation with *ktools* and reading *ktools'* output back into CLIMADA. The result is compared with the computation of the metric *at_event* of CLIMADA. To make all impacts of all events for every model visible in one plot, they are summed together:

$$x = \sum_i x_i, \tag{3.1}$$

where x is the overall impact and x_i is the impact at event i.

Since the error in translation stems from the discretization, the number and size of intensity and damage bins play a key role, as can be seen in Figure 3.1. To show this behaviour of the discretization, the hazard data "IBTrACS

from 1990 to 2004 over Florida with 2500 centroids" of CLIMADA's demo example is used.

To analyze the results with the given metric and example, the relative error is calculated:

$$r = \frac{x^{ktools}}{\sum_i |x_i^{ktools} - x_i^{CLIMADA}|}, \tag{3.2}$$

where r is the relative error, x^{ktools} is the overall impact for the ktools computation, x_i^{ktools} and $x_i^{CLIMADA}$ are, respectively, the impact computed by *ktools* and CLIMADA for every event i.

As can be seen in the log-log plot the relative error r of *ktools* is very high for a small number of bins. If the number of bins is increased and therefore the bins get smaller the error decays exponentially with a factor -1.4. This makes sense, since more and finer bins lead to a more accurate translation of the CLIMADA impact function to *ktools*.

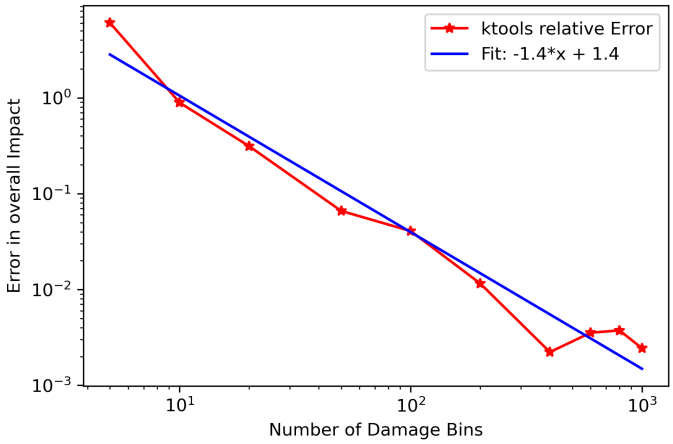

Figure 3.1: Log-log plot shows the *ktools* error in overall impact as a function of number of damage bins. The relative error of *ktools* to CLIMADA in overall impact is decaying exponentially with factor -1.4.

3.2 Uncertainty Calculation

The uncertainty calculation offered by *ktools* is one advantage for the CLIMADA users. To show an example usage of that tool, a toy example with the damage bins and impact function from Figure 2.2 is built. This corresponds to an impact function of *ktools* and therefore no discretization of the impact function is necessary. Since this example only includes a *ktools* impact function, no CLIMADA results can be shown. Further, three intensity bins are chosen, as in the example from Figure 2.2. Three events and four

exposures are generated in CLIMADA and translated to *ktools* with the help of Subsection 2.3.3.

For assessing the uncertainty of the impact, *ktools* needs the probabilities p^f and p^v explained in Subsection 2.2.3. p^f is chosen to be 0.7 for two events and 1.0 for the remaining event, whereas p^v is set to 0.8 for all damage bins.

As a comparison metric the impact per event is chosen. The results of this uncertainty calculation example are shown in Figure 3.2. One can clearly see that event 2 had probability p^f set to 1.0 as the standard deviation is smaller than the other two. This event has spread because of the uncertainty in the damage bins.

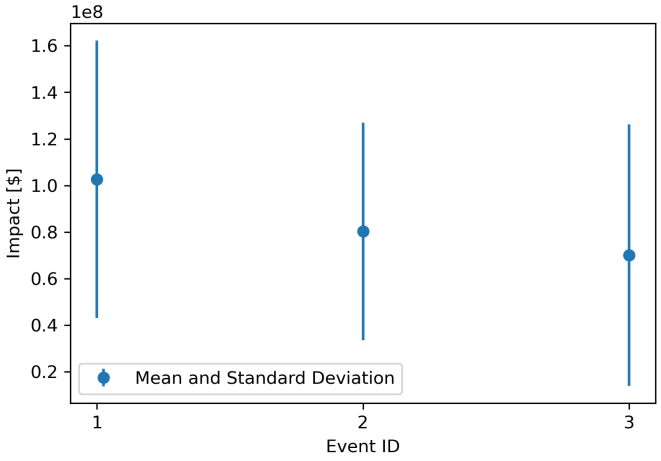

Figure 3.2: Mean and standard deviation of impact per event computed by *ktools*. As input a *ktools* impact function was given with three events and four exposures.

Chapter 4

Discussion and Outlook

The goal of this term paper was to achieve a data translation from CLIMADA to *ktools* and to perform computations with the translated CLIMADA data on *ktools*. Those two tasks are first discussed separately and afterwards an outlook in achieving a higher degree of interoperability is given.

4.1 Data Translation

The two frameworks which we coupled were built completely differently. Some translations between the two frameworks (e.g. impact function) were hard to achieve and required a good understanding of both frameworks. Unsolved translations remain because not every CLIMADA attribute has a correspondence in *ktools*.

From the results of the examples performed in Section 3.1 we conclude that translating data with restrictions from CLIMADA to *ktools* is possible. The accuracy it can be translated with is satisfying but not perfect. The limiting factor is the translation of the impact function due to the required discretization step.

4.2 Uncertainty Calculation

The other part we defined as interoperability is the use of *ktools* to perform uncertainty calculations with CLIMADA data. A result of such an example was shown in Section 3.2. This shows that it is possible to calculate an uncertainty corresponding to a computed loss. By using the implemented connection between CLIMADA and *ktools*, CLIMADA users are able to not only compute a loss but also its corresponding standard deviation.

Calculating the loss and its standard deviation requires a probability of every damage and intensity. If the simulation is small this is not a problem, as

in Section 3.2. But if a CLIMADA impact function would be discretized into many fine bins for an accurate translation, inserting these probability values gets cumbersome. Therefore, it is not recommended to perform a data translation and an uncertainty calculation at the same time.

4.3 Towards full Interoperability

This term paper lays the basis in interoperability. Translating CLIMADA data is possible and the uncertainty calculation can be used, both with restrictions however. Whether our developed implementations are integrated into the next CLIMADA release is still open for discussion.

A next step in data translation would be to make specific data sets of CLIMADA accessible for Oasis LMF users for a more complete translation. The translation of Oasis LMF data sets to CLIMADA which was not considered so far, would also be a preferable step in the future. This would lead to more available data for the CLIMADA users' models.

To achieve a higher degree of interoperability, on the one hand the coupling of CLIMADA to Oasis LMF itself could be done. The planned Oasis LMF migration to a Python implementation would clearly simplify that. This stronger coupling would offer more inference tools from Oasis LMF to CLIMADA users. On the other hand, more meetings with developers of both frameworks could result in finding more correspondences in hazard properties than could be found so far. This would enable a more complete data transfer and thus a higher degree in interoperability.

Appendix A

Appendix

A.1 ktools Data Files

Here an example of every file explained in Section 2.2.3 is shown.

damage bin dictionary

bin_index	bin_from	bin_to	interpolation	interval_type
1	0	0.4	0.2	1
1	0.4	1.0	0.7	1

Table A.1: Example of damage bin dictionary file to *ktools*' impact function of Figure 2.2.

footprint

event_id	areaperil_id	intensity_bin_index	prob
1	1	1	0.6
1	1	2	0.5
1	1	3	0.7
1	2	1	0.9
1	2	2	0.3
1	2	3	0.5

Table A.2: Example of footprint file with one event, two area perils, three intensity bins and its corresponding probability.

vulnerability

vulnerability_id	intensity_bin_index	damage_bin_index	prob
1	1	1	0.9
1	1	2	0.1
1	2	1	0.8
1	2	2	0.2
1	3	1	0.7
1	3	2	0.3

Table A.3: Example of vulnerability file to *ktools* impact function of Figure 2.2.

coverages

coverage_id	total_insured_value
1	10000
2	3000000
3	230900

Table A.4: Example of coverage file with three coverages and its corresponding total insured values.

events

event_id
1
2

Table A.5: Example of event file with two events.

items

item_id	coverage_id	areaperil_id	vulnerability_id	group_id
1	1	1	1	1
2	2	2	1	2
3	3	3	1	3
4	4	4	1	4

Table A.6: Example of items file which defines four items. Since all have different group ids, the items are not in the same correlation group when sampling.

gul summary xref

coverage_id	summary_id	summaryset_id
1	1	1
2	1	1
3	2	1
4	2	1

Table A.7: Example of gul summary xref file which includes four coverages. In this example the coverage one and two are grouped into summary_id one. If every coverage_id has a separate summary_id no summing is performed.

occurrence

event_id	period_no	occ_date_id
1	305	709
2	3468	28626

Table A.8: Example of occurrence file which includes two events. These are given with its period number it occurs in an the amount of days relative to the base date of 0/1/1900.

Bibliography

[1] G. Aznar-Siguan and D. N. Bresch. Climada v1: a global weather and climate risk assessment platform. *Geoscientific Model Development*, 12(7):3085–3097, 2019.

[2] Ben Hayes. Presentation "Oasis under the Hood". Access to content allowed.

[3] Oasis LMF. Documentation ktools. https://oasislmf.github.io/_downloads/9475f7bea0ce310a112ed3ca26e11e9b/ktools.pdf, Sidst set 03/11/2020.

[4] Oasis LMF, 2021. https://oasislmf.org, Sidst set 19/03/2021.

[5] Cambridge University, 2021. https://dictionary.cambridge.org/dictionary/english/interoperability, Sidst set 19/03/2021.