IV. Future models

The field site of this chapter lies in the realm of the digital and deals with technological entanglements in regional impact modeling. Regional climate modeling is an intriguing scientific practice, because it seems to reverse climate science's obsession with the global scale. As Paul Edwards has shown, meteorologists have fought for centuries to "make global data" (building a global observation and communication infrastructure) and to "make data global" (standardize heterogeneous datasets) (2010). This infrastructural work was a major achievement and enabled the discovery and scientific proof of climate change. What are the reasons for the renewed interest in making global data local again or, as Mahony (2017) puts it, "the (re)emergence of regional climate"?

Global-to-local

From the perspective of simulation modelers, regionalization is mainly a matter of resolution. We can illustrate both these concerns by a description of RCMs on the website of the CORDEX[44] project, a globally coordinated effort in downscaling global models to regional scales:

> Global Climate Models (GCM) can provide us with projections of how the climate of the earth may change in the future. These results are the main motivation for the international community to take decisions on climate

44 Coordinated Regional Climate Downscaling Experiment.

change mitigation. However, the impacts of a changing climate, and the adaptation strategies required to deal with them, will occur on more regional and national scales. This is where Regional Climate Downscaling (RCD) has an important role to play by providing projections with much greater detail and more accurate representation of localised extreme events.[45]

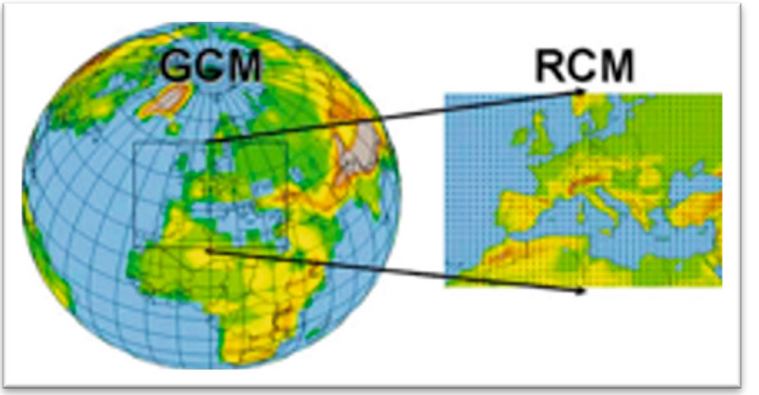


Figure 32: An RCM domain embedded in a GCM grid.
Image source: Giorgi (2008)

There are three established methods to create local climate data: (1) Increasing the resolution of a global model, (2) running a statistical regional model, which derives local data from the output of a global model, or (3) setting up a dynamic local model with its own physical logic (Mahony 2017: 140ff). All of these methods have their strengths and weaknesses, which then also define possible application fields. Regarding their representational logic, the simulation of regional climate is different from global modeling, in the sense that regional models must represent phenomena, such as high- and low-pressure areas, vegetation, land use, glaciers and snow cover. And, as climate scientists often argue, regional models must be able to simulate in a higher

45 https://www.cordex.org/about/what-is-regional-downscaling/, retrieved on May 5, 2019.

resolution than global models in order to represent these processes accurately. Meteorological models of weather prediction need to perform similar tasks and enable verification regarding empirical observations; regional modeling builds mainly on model structures in meteorology (Jacob et al. 2017: 29). We may briefly compare two of these methods – statistical downscaling (2) and dynamic downscaling (3) – to illustrate some of these differences.

Dynamic downscaling

Models of dynamic regionalization calculate climate impacts with a three-dimensional excerpt of the atmosphere; this is similar to the global models but with a higher resolution. Dynamic models basically resolve a theoretical system of equations on a defined spatiotemporal grid. The equations represent laws of conservation for energy, impulse and the mass of the air, as well as water and water vapor. The dynamic regional model starts with the outputs of a global model and obtains new boundary values from the latter every six hours (in simulated time). Consequently, the global model is also formative for the long-term variability and the large-scale processes of the region of the model (e.g. Europe). The regional climate is then calculated gradually by increasing the spatial resolution of the model: Firstly, to a grid mesh width of 50 km and then down to 10, 7, 3 or even 1 km. The higher spatial resolution enables the representation of characteristics of the Earth's surface, such as the altitude structure and land cover, and processes, such as local precipitation and cloud coverage. Dynamic modeling is often described as the "royal road" (Orlowski 2007: 3).

Statistical downscaling

A statistical regional model works differently. Statistical modeling explores relationships between large-scale weather conditions or global circulation patterns and local climate data. The statistical model STARS, for example, developed at the Potsdam Institute, rearranges the time series of climate variables observed and simulated in order to take

into account prescribed, linear trends. The results are synthetic, comparable time series of meteorological variables at the places of meteorological weather stations. As a result, the resolution is determined by the spatial density of weather stations. The advantage of statistical models is that they need a lot less computing time than the dynamic models. However, they are unable to simulate events that are fundamentally different or more extreme than those observed in the past because they are literally mirrors and conditioned projections of the past into the future (translated and summarized from Jacob et al. 2017: 28f). The PIK's statistical model STARS has triggered a number of scientific controversies due to these representational flaws. The model has been operationalized to simulate future climate change in Germany, making statements about developments at the spatial scale of counties and districts. Further down the model chain, STARS has been used to drive a number of climate impact models which simulate the consequences of precipitation and water systems (floods, extreme weather), agriculture (drought, flowering times, cultivation of new wine grapes), tourism and health-related issues (heatdays). The simulation outputs triggered a variety of discussions in the mass media[46] and provided a scientific base for interdisciplinary studies making sense of the future with climate change in Germany (Gerstengarbe et al. 2013). Ten years later, scientists at the PIK challenged the mathematical logic of the STARS model and some interpretations of the simulation outputs (Wechsung/Wechsung 2016, 2015). Inter alia, the new reconsiderations have been possible thanks to experiments migrating STARS to other regions, such as the Chinese Guanting region (Wechsung/Schellnhuber 2018). Myanna Lahsen showed that modelers tend to be very protective of their own models, given the long time they spend 'raising' them (2005).

46 See, for example, https://www.welt.de/wissenschaft/umwelt/article5456480/Wie-der-Klimawandel-Deutschland-trifft.html.

However, the episode concerning STARS also exemplifies the sophisticated self-correcting mechanisms in place within the climate sciences.

The main strategy of climate scientists to account for the uncertainties of different models and their procedures is to compare and average them within model ensembles, similar to that in global climate modeling practice. Coordinated efforts, such as CORDEX, then produce coordinated sets of regional downscaled projections for all the regions of the world. As a result of the ensemble process, scientists engaging in further modeling at the local level (e.g. climate change impact modeling) do not have to bother with choosing between different models or their outputs but can rely on standardized climate time series data that drive their own predictions of the future. However, this also means that impact modelers have to trust the soundness of the original downscaling models, simulation process, averaging methods and data output mechanisms. Paul Edwards has shown that the *vast machine* of climate science is the extraordinary instance of trusted infrastructure.

Place-to-place

As Martin Mahony has shown, regional modeling can be described as a practice of translation. For him, RCMs broadly fulfill two functions: On the one hand, they are employed to "re-invest the global climate with some of the local meaning of which it is stripped in the moment of its construction" (Mahony 2017: 140). On the other hand, RCMs are key tools for nation-states translating climate change into something they can govern:

> National maps of climate impacts re-territorialize climate change, and enable states to perform a competent engagement with risks and uncertainties that are paradoxically beyond their own capacities of control. (ibid.)

Once a regional model is technically developed, the question arises to what extent it is location-specific or if it can be easily operationalized for multiple geographic spaces. In other words, the question is how much the map (model) resembles a territory. The human geographers Mike Hulme and Martin Mahony have investigated such questions of mobility and mutability of local climate models in a number of articles (Hulme 2008; Mahony 2017; Mahony/Hulme 2012). As Mike Hulme has put it, climates do not seem to travel well between scales:

> It is important to notice what happens in this circuit of transportation. Weather is first captured locally and quantified, then transported and aggregated into regional and global indicators. These indicators are abstracted and simulated in models before being delivered back to their starting places (locales) in new predictive and sterilised forms. 'Digitised' weather for virtual places can even be conjured from these models using stochastic weather generators. Through this circuitry, weather – and its collective noun climate – becomes detached from its original human and cultural setting. (2008: 7)

It appears convenient to investigate issues of scale and mobility in the context of regional climate modeling, considering that such geographic mobilization is often an explicit goal of these research endeavors. Can the German model be migrated to China, the Elbe model to the Yangtze river, as a movement from one represented Euclidian space to another? As a side note, it may be added that there have also been attempts to apply global climate Earth models to other planets in order to learn about their climates and about ours (Kasting et al. 1988) The prime case is the investigation of the Venus syndrome (Goldblatt/Watson 2012), as a model for 'climates gone bad' or 'runaway climate change.' Nevertheless, the most obvious practical case for climate-model migration is from one geographic place to another.

Mahony and Hulme aim at assessing, "how scientific tools are able to overcome the friction of distance and attain 'usefulness' in new places, and the effects of these transfers on the epistemic landscapes of their new environments" in their investigation of the PRECIS (Producing Regional Climates for Climate Impacts Studies) model (2012: 198). The latter is an RCM developed by the United Kingdom's Met Office Hadley Centre. According to the Hadley Centre's website, it is

> [...] a regional climate model (RCM) that takes large scale atmospheric and ocean conditions from observations or global climate models (GCM) where horizontal resolutions vary from 100 to 300 km, and downscales it over a region of interest to resolutions of 25 or 50km. This allows for a more realistic representation of the climate over the region of interest, accounting for complex surface features such as mountains, coastlines and islands which are not resolved in the global models. [47]

The PRECIS is an accustomed traveler. While originally conceptualized for European territory, the system has migrated to a variety of places, including India and South Africa. It has not only toured throughout the world but has also been in the hands of a variety of different actors. Building on Anselm Strauss' (1978) social worlds concept, Mahony and Hulme argue that PRECIS facilitates interaction and exchange between multiple worlds and sub-worlds:

> [...] PRECIS can be seen to facilitate interaction and exchange between a number of worlds and sub-worlds: model developers, the climate impacts community, global and national political assemblages, non-governmental institutions etc. The climate arena, within which the various actors interact, provides a transaction space [...] whereby asymmetrical relationships of dependency can develop at institutional and disciplinary boundaries. (Mahony/Hulme 2012: 208)

47 https://www.metoffice.gov.uk/research/applied/international/precis/introduction.

The authors also characterize PRECIS as a *boundary object,* building on Susan Star and James Griesemer's (1989) conceptualization:

> PRECIS' multivalent purposes, as articulated by developers, partners and users, its notional flexibility engendered by its mobility, and its ability to fulfill a range of substantive, instrumental and discursive demands make it eligible for this description. (ibid.: 208)

As multifarious as RCMs may individually behave as politico-scientific devices, they are gradually becoming *obligatory passage points* (Callon 1984) for the accomplishment of a political sagacity (Mahony/Hulme 2012 208).

> This is achieved through the translation of instrumental goals and the deployment of normative discourses of vulnerability and scientific realism, the consequence being a community pursuing knowledge that possesses high spatial resolution and precision. This pursuit is facilitated by the rendering of planned adaptation as captive to, or an ancillary of, the ability to predict future climatic changes on the scales that most interest decisionmakers. (ibid.)

Mahony and Hulme highlight a number of consequences of the establishment of RCMs as obligatory passage points. Notably, the recurrence to (a certain type of) models may privilege some approaches of climate adaption strategies over others (optimal, rather than robust) (Dessai/Hulme 2007). However, Mahony and Hulme also draw more general conclusions from such preferential treatment of model predictions over other practices to think about the future. For these authors, they represent an "unfolding geography of epistemic power" with climate as a "chief determinant of humanity's putative social futures" (Hulme 2011; Mahony/Hulme 2012).

Such climate determinism and reductionism are also considered and debated in the community of climate impact modelers at the PIK. Many impact modelers are not trained as climatologists but as economists,

agriculture specialists and hydrologists. As has been mentioned previously, they have to rely on climatologists to drive their own projections of the future:

> TC: Yes, so the problem is, we usually only see the future from a climate perspective. There are very good climate models now, compared to the ones we had a few years ago. […] So, we have a variety of scenarios that we can feed into it as different realizations of future climate. And we just look at how the [water] runoff behaves. However, change in land use is much more difficult to describe. So we haven't done it for this area. (Interview with Torsten Casius, translated by the author)

In this sense, climate models do not only project future climates but also imprint a view of other future developments, such as land use changes and urbanization. Put positively, climate change is also a mobilizing element for other environmental knowledge, then traveling into the future as a free rider. Climate science provides a spatiotemporal grid for the future, which can then be colored by impact modelers and other actors. It might, therefore, be understandable that impact modelers often maintain a controversial relationship with climate models and modelers.

Infrastructural migration

The existing literature in STS has treated regional and impact modeling as a matter of knowledge translation raising representational issues. This equally includes considerations of epistemological and political representation.

> […] scientists, campaigners, and politicians have long been aware of the politically paralyzing effects of knowledge claims that refer to abstract, global realities rather than the local realities of everyday existence or routine political decision making. (Mahony 2017: 139)

By contrast, model migration can also be understood as a technological issue, migrating a model from one machine, system or infrastructure to another. In one instance, Martin Mahony briefly discusses the computational practices and infrastructures making models travel. Common issues of epistemic uncertainty and model opacity in regional modeling are usually countered by choosing open-source software tools. However, Mahony claims that the trend toward open-source simulation software and models, accompanied by a rhetoric of transparency and reflexivity, is, in fact, characterized by a high degree of epistemic opacity. According to him, simulation modelers often do not truly understand the design principles and assumptions of the simulations and models at stake. This *epistemic opacity* of modeling technology (Kouw 2010: 4) is strongly dependent on factors such as the detail of the accompanying handbooks and the user's trust in the scientific credibility of the model constructors (Mahony 2017: 152). Drawing on Matthijs Kouw's extended interpretation of the term 'vulnerability' [48] (2010: 1) and his analysis of modeling technology in hydrology, Mahony argues that epistemic opacity can create a kind of vulnerability based on software design and use:

> In the case of PRECIS, this "epistemic opacity" was a product both of the desire to produce a usable tool, and of the wish to preserve the authority of the Hadley Centre's own development and coding. PRECIS travels the world through a network of national contact points who receive training from the

48 Vulnerability is a key term in climate impact research. While many contesting definitions of the concept exist (Füssel 2005), the IPCC characterizes vulnerability as follows:

"The degree to which a system is susceptible to, or unable to cope with, adverse effects of climate change, including climate variability and extremes. Vulnerability is a function of the character, magnitude, and rate of climate variation to which a system is exposed, its sensitivity, and its adaptive capacity." (McCarthy/IPCC 2001: 6).

> Hadley Centre with support from governmental agencies. For one scientist not associated with the Indian contact point in Pune, PRECIS has been an inaccessible tool despite arguments that the model should be run at more than one location in India [...]. (Mahony 2017: 152)

Epistemic opacity in regional climate modeling for Mahony is then shaped by several factors, including software interfaces that hide the model's core code, expert considerations and restrictions regarding code access, and the material realities of scientists in resource-poor (and, thus, computationally limited) institutions and/or countries (ibid.:155). In this reading, the discourse of open software promises the mobilization of computer models, their stored knowledge and instrumental capacities to various geographic places, user communities and technical systems. However, these promises are only partially met for Mahony and fail to mitigate the epistemic opacity of computer models and modeling as a techno-scientific practice.

Against this view, I would argue that scientists indeed manage to address epistemic opacity quite successfully with a number of techniques that are discussed as follows.

Investigating a model

This second part of the chapter will address technological practices of mobilization in climate impact modeling, which I will refer to as *mobile modeling*. I will discuss these recent practices in impact modeling using the empirical example of CLIMADA (CLIMate ADAptation), a climate risk assessment and damage calculation model, tool and platform. The software stack and infrastructural entanglement around CLIMADA can be exemplary for shifting practices in climate impact modeling and scientific software development in general. I came across CLIMADA while interviewing Tobias Geiger, a PIK expert for the modeling of extreme weather events, hurricanes and particularly their economic damages. Tom is not the main developer of CLIMADA but a contributor to its code, where he added a module for the simulation of hurricanes and

their damage caused to local economies. The model itself (CLIMADA) has been developed and maintained by David Bresch, Professor for Weather and Climate Risks at the Swiss Federal Institute of Technology in Zurich (ETHZ). The user manual of the model states that "CLIMADA is an open-source and -access global probabilistic risk modelling and adaptation economics platform." (CLIMADA Manual: 1) It aims at strengthening weather and climate-resilient development and providing decision-makers "with a fact base to understand the impact of weather and climate on their economies, including cost/benefit perspectives on specific risk reduction measures" (ibid.). The functionalities of CLIMADA can be illustrated by two visualizations from that part of the model. Based on spatiotemporal data representing the distribution of 'assets' (e.g. buildings, agricultural areas) at a geographic location, CLIMADA simulates the economic damage of natural disasters to these entities. The map in Figure 34 depicts the risk exposure of assets as green spots (existing but low exposure) and a few red ones (high exposure). One can specify different types of disasters (e.g. a hurricane or a Tsunami), time frames or territories by manipulating input data and functions of the model.

CLIMADA can represent historic events or simulate future ones, the latter based on projections of socioeconomic and climate variables. It also makes a prediction about the share of additional damage caused by climate change, with Figure 34 showing a possible result of these calculations on the right. It prognosticates an accumulated risk of damage of about 34 billion USD until the year 2040. It also anticipates that a large faction of this damage (21 billion USD) will be attributable to climate change.

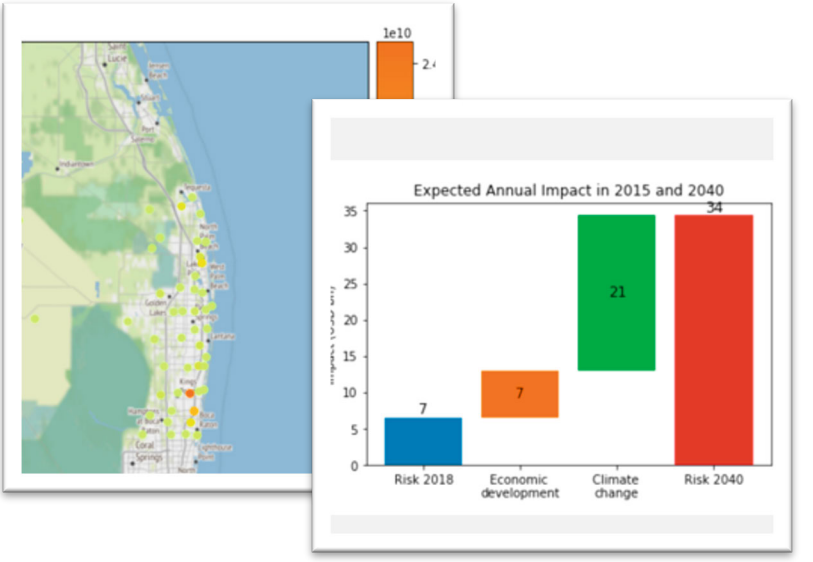


Figure 33: Mapping exposures to natural hazard at a location within the US Federal State of Florida. Source: CLIMADA Jupyter Notebook

Access log

I developed the methodological device of an *access log* to trace the infrastructural elements of CLIMADA. The term is inspired by the use of the word in ship navigation and software development. A logbook or log in shipping is a record of important events in the management, operation and navigation of a ship[49] Software development draws from this use of the word in shipping and translates it to the situation on the web:

> A Web log file records activity information when a Web user submits a request to a Web Server. The main source of raw data is the web access log which we shall refer to as log file. As log files are originally meant for

49 https://en.wikipedia.org/wiki/Logbook, retrieved on April 3, 2019.

> debugging purposes.
> (Suneetha/Krishnamoorthi 2009: 327f)

In my case, the access log is a special 'field-note' that traces the steps to set up the model on a local machine (laptop).

- open github.com website
- type 'CLIMADA' into the search field
- open the project page
- tap 'clone or download'
- wait 15 seconds (file is downloading from github.com to my laptop)
- click on the zip-file 'climada_python-master.zip' in the downloads folder of my laptop (file unzips)
- copy CLIMADA folder into Python folder structure on my laptop
- open Anaconda Navigator (click on application alias) on laptop and launch the anaconda IDE (integrated development environment)
- Launch Python Jupyter notebook
- Terminal opens and launches Python on laptop
- Open readme file in notebook
- Read installation instructions
- Open linked guide[50] for more info
- As indicated in guide, install dependencies in Anaconda by choosing Environments/Import
- Anaconda creates a new software environment for CLIMADA (takes 5 min)
- In Jupyter notebook, navigate to climada_python-x.y.z. repository and open doc/tutorial/1_main_climada.ipynb file
- Jupyter launches CLIMADA notebook in browser (Firefox)

50 https://climada-python.readthedocs.io/en/stable/guide/install.html, retrieved on May 6, 2019.

The access log helped me to identify infrastructural elements and relationships within and toward technologies of interest. It helped, for example, to identify Github, Python, Anaconda, specific libraries and Jupyter Notebook as entities to be considered in my research. One could characterize these elements as 'dependencies'[51] of a specific technology or infrastructure.

This entire process of setting up CLIMADA takes about 30 min on an Apple Macbook Pro (2017 model) laptop. It could equally be installed on a Microsoft or Linux machine taking the same steps. Of course, a successful setup also comes with sociotechnical preconditions. One had to learn Python, getting to know Jupyter Notebooks, installing Anaconda, Python, and its libraries on the local machine. One needs a functioning internet connection. It is useful to know where to find things on Github and to tab communities at Stack Overflow for troubleshooting. However, it is still impressive that formerly highly esoteric technologies, such as climate models, are, or appear at least, relatively open and accessible with broadly disseminated (programming) skills. In the following, I will focus on some of the elements and relationships identified in the access log, as they particularly characterize contemporary scientific programming and *mobile modeling* especially.

Coding openness

The CLIMADA was originally developed in MATLAB,[52] a proprietary program owned by the US-American company *Mathworks,* which specializes in mathematical computing software. However, in 2017, it was

51 This use of the word differs from the one in computer science. Here, dependencies are literally the external pieces of code that have to be called by a specific program.

52 The official website of MATLAB, retrieved on June 4, 2019, via https://www.mathworks.com/products/matlab.html.

decided to translate the whole CLIMADA code into the Python programming language and to make it available as open-source and free software. The exiting MATLAB version, by contrast, will no longer be maintained.[53] 'Open-source' and/or 'free' means that software source code is equipped with a specific legal license, in this case, a GNU lgpl (Lesser General Public License).[54] It is beyond the scope of this study to discuss the different versions of open software licenses, even if this licensing choice has a strong impact on the meaning of 'openness' and the politics of amplification in question. In a historical perspective, it must be said that opening programming code is not a new practice in software development but goes back to the origins of the craft in the 1960s and 1970s, as described by von Hippel and von Krogh:

> In the early days of computer programming commercial "packaged" software was a rarity – if you wanted a particular program for a particular purpose you typically wrote the code yourself or hired it done. Much of the software development in the 1960's and 1970's was carried out in academic and corporate laboratories by scientists and engineers. These individuals found it a normal part of their research culture to freely give and exchange software they had written, to modify and build upon each other's software both individually and collaboratively, and to freely give out their modifications in turn. This communal behavior became a central feature of "hacker culture." (2003: 3f)

53 Information from CLIMADA Github page, retrieved on June 6, 2019, via https://github.com/davidnbresch/climada.

54 Wikipedia page of the GNU LPGL license agreement:

"The license allows developers and companies to use and integrate a software component released under the LGPL into their own (even proprietary) software without being required by the terms of a strong copyleft license to release the source code of their own components. However, any developer who modifies an LGPL-covered component is required to make their modified version available under the same LGPL license." Retrieved on June 4, 2019, via https://en.wikipedia.org/wiki/GNU_Lesser_General_Public_License.

Such practices have come with a promise that everyone can potentially use the software, which is also true for the CLIMADA collaboration between ETHZ and PIK researchers:

> It's just that the whole basic structure exists in MATLAB. Which, in my opinion, is not very user-friendly. Because, on the one hand, there are license fees. And, on the other hand, it is not so user-friendly for me. Mmm. But yes, that will perhaps also change in the future. So, there are already the ideas that maybe you can import this complete package into Python and then use it there. (Interview Geiger, translated by the author)

The interview with Geiger was carried out at the beginning of the cooperation with the ETHZ. Two years later, the whole code had been translated from MATLAB to Python. In this sense, the practices of 'mobilization' of CLIMADA had been successful. Initially, opening the source code triggered an engagement of the PIK researchers to contribute to the ETHZ software. On the other hand, embedding the PIK module in a broader modeling endeavor (CLIMADA) also amplified the impact of the work carried out in Potsdam.

As this example shows, 'coding openness'[55] is more than a license issue by far. It includes a variety of ideas, practices and infrastructures, some of which are discussed further below.

Pythonization

The shift from proprietary to open-source programming languages and environments translates particularly to a transformation we might refer to as *Pythonization.* Scientific models are increasingly imagined and formulated in Python, a high-level language such as C, C++, Perl and

55 Term taken from Prof. Dr. Claudia Müller-Birn's course taught at the Computer Science department of Freie Universität Berlin. More info on https://www.mi.fu-berlin.de/en/inf/groups/hcc/teaching/summer_term_2019/coding-openness.html, retrieved on June 5, 2019.

Java. As computers can only directly read low-level languages ('machine' or 'assembly languages'), programs written in a high-level language have to be processed before they can run. This extra processing takes some time, which is a disadvantage of high-level languages. However, the latter also carry enormous advantages. *Think Python: How to Think Like a Computer Scientist*, a popular textbook introducing the Python programming language to prospective users, puts it as follows.

> First, it is much easier to program in a high-level language. Programs written in a high-level language take less time to write, they are shorter and easier to read, and they are more likely to be correct. Second, high-level languages are portable, meaning that they can run on different kinds of computers with few or no modifications. Low-level programs can run on only one kind of computer and have to be rewritten to run on another. (Downey 2012: 1)

We will come back to this feature of 'portability' in Chapter V. While this explanation clearly shows the advantage of high-level programming languages in comparison to assembly languages, it does not explain why Python is currently thwarting formerly dominant languages, such as Java or C++. We can find some arguments for its success in another Python reference textbook, *A Whirlwind Tour of Python* published in 2016. According to its author, Jake VanderPlas, "the appeal of Python is in its simplicity and beauty, as well as the convenience of the large ecosystem of domain-specific tools that have been built on top of it" (2016: xii). As for this "simplicity" and "beauty," we can find a spiritual self-description of these qualities built right into the heart of the language code by typing the command "import this" into a Python console:

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Figure 34: The Zen of Python. Source: My own screenshot, Zen by Tim Peters

On a more concrete and technical level, one thing that distinguishes Python from other programming languages is that it is interpreted rather than compiled: "[T]his means that it is executed line by line, which allows programming to be interactive in a way that is not directly possible with compiled languages like Fortran, C, or Java" (ibid.: 5). Such interactive coding enables extensions, such as the *Jupyter Notebook,* which revolutionize the way scientific programming works (see more below).

The trend toward Python in coding practice was a recurring theme in my interviews and discussions with impact researchers. Referring to Gabriele Gramelsberger's characterization of FORTRAN as the *lingua franca* of climate modeling (2008a: 144), we may argue that Python is increasingly taking up this position in climate impact research. The reasons for this shift go beyond the mere consideration of Python's qualities as a programming language. They are linked more to an infrastructural entanglement that includes elements such as multipurpose software packages and powerful community platforms.

Packaging code

The large ecosystem of domain-specific tools for scientific computing and data science is built around a group of modules[56] and packages[57] (also referred to as 'libraries'[58] in everyday speech). Common Python packages are NumPy (storage and computation for multidimensional data arrays), SciPy (numerical tools, such as numerical integration and interpolation), Pandas (a set of methods to manipulate, filter, group and transform data), Matplotlib (an interface for the creation of publication-quality plots and figures), Scikit-Learn (a toolkit for machine learning) and IPython/Jupyter (for the creation of interactive, executable documents) (VanderPlas 2016: 1f). The packages can easily be imported into one's own programming code, where they perform a variety of tasks for the structuring, analysis and representation of data. As a result, Python has become particularly effective for coping with the contemporary challenges of data-intensive science, 'Big Data' or 'data science':

> As an astronomer focused on building and promoting the free open tools for data-intensive science, I've found Python to be a near-perfect fit for the types

56 "Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module; definitions from a module can be imported into other modules or into the main module (the collection of variables that you have access to in a script executed at the top level and in calculator mode)." From the Python documentation, retrieved via https://docs.python.org/3/tutorial/modules.html on July 6, 2019.

57 A package is a collection of modules. See Python documentation, retrieved via https://docs.python.org/3/tutorial/modules.html#packages on July 6, 2019.

58 In contrast to Java Script and other languages, Python does not formally entail 'libraries.' However, it is often used synonymously for modules and packages.

> of problems I face day to day, whether it's extracting meaning from large astronomical datasets, scraping and munging data sources from the Web, or automating day-to-day research tasks. (ibid.)

This aspect of automation also becomes increasingly important within climate impact research, as the following excerpt from an interview with a PIK scientist shows:

> So, the data comes from two sources, one HTML website (ratification) and a CSV (emissions). I wrote a Python script to extract the data. The export from the HTML page is automatic." (Interview Gatow)

While the majority of the work is still arranged around simulation models, impact researchers also increasingly engage in capturing live data, be it from the web or from updated servers providing satellite imagery. These practices pertain to what is now considered as *data-science* rather than computational science. In a loose understanding, the term data-science is often associated with the contemporary challenges of 'big data' and opportunities in machine learning, deep learning and artificial intelligence to deal with it. Conceptually, data science may grasp a more generalized shift in academia and industry to take 'data' as the primary object to be dealt with (Ribes 2018: 2), translating to an entanglement of data collection, engineering, analytics and representation (Computing Research Association 2016). Experts in the field have also highlighted the transdisciplinary nature of data science for academia, arguing that "across academic disciplines, the computational and deep data problems have major commonalities. If researchers across departments join forces, they can solve multiple real-world problems from different domains" (O'Neil/Schutt 2013: 15). A potential has also been seen particularly in 'AI for good,' with data science addressing specially socio-environmental challenges, such as climate change, biodiversity loss and natural risk prevention and management (Bundesregierung 2018: 17; International Telecommunication Union 2018:

26; Karpatne et al. 2017). Nevertheless, a variety of actors have also highlighted the potential risks of data science practices, linked to issues such as bias in and opacity of such algorithmic systems (AI Now 2018; Crawford 2013; Crawford/Calo 2016).

In any case, the aptitude for data-science explains some of the attractiveness of Python for actors outside the world of science:

> Conceived in the late 1980s as a teaching and scripting language, Python has since become an essential tool for many programmers, engineers, researchers, and data scientists across academia and industry. (VanderPlas 2016: 1)

Industrial actors building their products with Python include the big players of the data and platform economy, such as Google, Instagram, Spotify, Netflix, Uber, Dropbox, Pinterest and Reddit.

Wrapping code

It is not always possible or useful to rewrite the entire code of existing FORTRAN or MATLAB models in Python for obvious reasons, only to make it more accessible for potential others. This process of line-to-line translation might take months and could only be taken up by a researcher who is familiar with the underlying logic of the model in question and in both languages, FORTRAN and Python. Nevertheless, it may often be desirable or even necessary to preserve existent models programmed in (what we may call) esoteric and legacy computer languages[59] and make them compatible with contemporary software

59 It is important to make the difference here between legacy and esoteric programming languages. By 'legacy languages,' I understand programming languages that are no longer in use today, partly because of their incompatibility with current technical systems. By 'esoteric languages,' in contrast, I mean that only a few people have an expertise in programming them. Not all systems can be programmed in Python or other flexible languages. The

technologies and infrastructures. Developers can develop *wrappers,* which translate code from one language into another, to do so. An example of this tactic is *Pymagicc,* a Python interface for the FORTRAN-based climate model MAGICC (Model for the Assessment of Greenhouse-Gas Induced Climate Change; Meinshausen et al. 2011). The original MAGICC model is used by several modeling groups to assess the pathways of future emissions in climate policy analyses. The promise of Pymagicc is that it mobilizes MAGICC for actors that are not familiar with FORTRAN and the model-specific software environment. By contrast, Pymagicc runs on Windows, macOS and Linux and uses a standard tabular data structure (DataFrames from the Pandas library) for emissions scenarios. As a result, the MAGICC model parameters and emissions scenarios can be modified using Python, without having to touch the original FORTRAN model. Considering that such Python wrappers have recently been developed for a number of climate models, this practice also provides new opportunities for model comparison. The Pymagicc source code, documentation, an issue tracker and Jupyter Notebook are made available via a Github repository (see further below). The model can even be explored interactively in a web browser via the Binder project[60] (Gieseke et al. 2018: 1f).

Tapping Crowdknowledge

The use of Python by industrial actors also points to another ground for Python's increasing dominance: By choosing a particular entanglement

contemporary Machine Learning algorithms are all programmed using precise languages, such as C, rather than Python. But they typically come with a Python wrapper, which allows them to be accessible to a wider user community. An example is Googles' machine learning library Tensorflow, whose core runs on highly optimized C++, while providing direct manipulation via Python.

60 https://mybinder.org/v2/gh/openclimatedata/pymagicc/master?filepath=notebooks/Example.ipynb.

of technologies (e.g. Python, common packages) and way of doing things (open sourcing code), scientists become part of an ever-exploding community of practice (Lave 1991) gathering around terms such as 'data science,' 'data analytics' and 'machine learning.' As VanderPlas highlights, "[...] if there is a scientific or data analysis task you want to perform, chances are someone has written a package that will do it for you" (2016: 2). On the one hand, this means that developers will easily find ready-to-use code snippets solving all sorts of problems encountered by others which can be integrated into ones' own programming code. On the other hand, the web and its dedicated platforms provide millions of troubleshooting tips for coding issues. While there are multiple platforms on the web providing such services, the most prominent ones are github.com, stackoverflow.com and medium.com.

Github is an American company and community platform providing a variety of services for software development, including hosting, distributed version control (Git[61]) and source code management. It also provides access control and several collaboration features, such as bug tracking, feature requests, task management and wikis for every project. Github is a subsidiary of Microsoft, which acquired the company in 2018. GitHub offers plans for free, and professional and enterprise accounts, and its free accounts are commonly used to host open-source projects. We can take the example of the CLIMADA profile (see Fig. 35) to describe the structure of content on Github. The main view of a repository[62] on Github shows the project title, buttons for community functions (watch/star/fork), a number of content structuring menus (code, issues, pull requests, projects, wiki, security and insights), statistics (1249 commits, 2 branches, 14 releases, 10 contributors, license info), the file and folder structure of the project, and the text of the

61 More info on https://git-scm.com/, retrieved on June 2, 2019.

62 https://github.com/CLIMADA-project/climada_python, retrieved on June 2, 2019.

readme file (including installation instructions and project documentation. One can easily download the packaged code ('clone or download') or contribute to the project (e.g. 'create new file,' 'upload files').

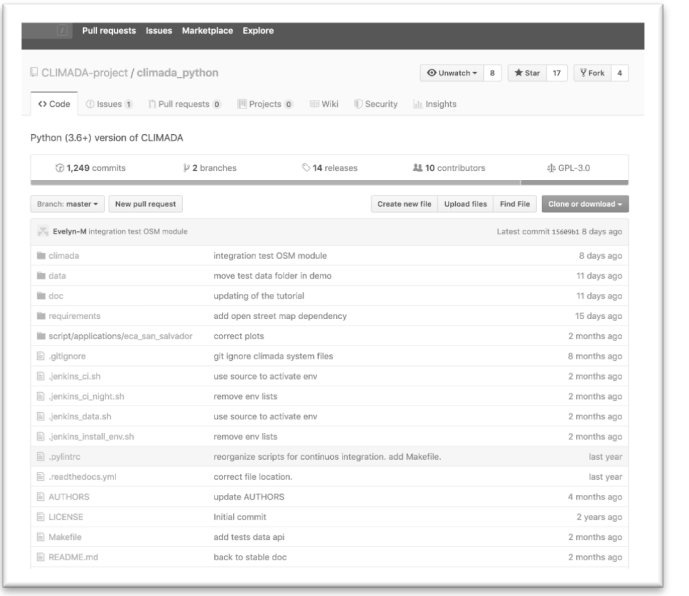


Figure 35: View of the CLIMADA repository[63] on Github.
Source: My own screenshot

On a technical level, the key feature of Github is the underlying technology *Git*, a free and open-source distributed control system version, which is commonly used nowadays in collaborative software development. With Git, every change to a software code becomes perfectly accountable and traceable for others. 'Others' here can mean the members of a well-defined and closed community[64] (e.g. a company, network or

63 https://github.com/CLIMADA-project/climada_python, retrieved on June 5, 2019.

64 E.g. through deployments of Git, such as gitlab. See http://about.gitlab.com, retrieved on June 5, 2019.

organization) or – as in the case of Github – the World Wide Web. Provided the legal preconditions (an open software license) have been met, developers can also create ('fork') and develop their own version of an existing software project on Github, making this provenance and genealogy perfectly visible and traceable. On the other hand, Github is essentially a social network connecting developers and users of software projects. Similar to Stack Overflow, Github enables software developers to publicly ask questions related to programming code and infrastructure. In contrast to Stack Overflow (see below), the discussions are happening around a particular project, its developers and community.

Stack Overflow is a privately-held community platform for programmers and businesses, featuring questions and answers on a wide range of topics in computer programming. Users find these answers either by entering questions into the search console of the platform or by Googling it and being forwarded to the site. The answers are rated by the community for their usefulness and ranked accordingly by the platform. In the rare cases where an answer to a question is not available in the archive, one can open a new thread and ask the community for help. As a result, stack Overflow facilitates a nearly perfect information clearance between questions and answers to programming issues. The more common a problem, programming language and technological stack (e.g. data issues in Python), the greater the chance that an answer can be found on Stack Overflow.

While Github gathers actors around a specific code and Stack Overflow provides solutions to concrete coding problems, *medium.com* is the platform to negotiate more conceptual issues linked to software development. Medium is an online publishing venture launched in August 2012. The Wikipedia article of the platform describes it as "an example of social journalism, having a hybrid collection of amateur and professional people and publications, or exclusive blogs or publishers on

Medium, and is regularly regarded as a blog host."[65] Initiated by a co-founder of Twitter (Evan Williams), the original idea for Medium was to provide a way to publish writing and documents longer than Twitter's 140-character (now 280-character) maximum. Its self-image is organized around promises of innovation, fresh ideas and creativeness:

> Ideas and perspectives you won't find anywhere else.
> Medium taps into the brains of the world's most insightful writers, thinkers, and storytellers to bring you the smartest takes on topics that matter. So whatever your interest, you can always find fresh thinking and unique perspectives.[66]

Issues discussed on Medium are not limited to technology but the software development and startup communities are among the most active ones on the platform. Entering 'why learn python' in Medium's search console returns a multitude of articles, one published within the influential social blog *Hackernoon* titled *10 Reasons to Learn Python in 2019*. The items on the list confirm many of the aspects discussed within this chapter and include "Data science, Machine Learning, Web development, Simplicity, Huge community, Libraries and frameworks, Automation, Multipurpose, Jobs and Growth, and Salary."[67]

With the rise of community platforms, such as Github, Stack Overflow and Medium, programming code and programming knowledge has become increasingly distributed between different spaces, actors and artifacts. As Adrian Mackenzie has highlighted earlier, "[…] software has hybridized itself wildly with other media and practices and is likely

65 See Wikipedia's entry for medium.com at https://en.wikipedia.org/wiki/Medium_(website), retrieved on April 2, 2019.

66 See https://medium.com/about, retrieved on April 2, 2019.

67 See https://hackernoon.com/10-reasons-to-learn-python-in-2018-f473dc35e2ee, retrieved on April 2, 2019.

to continue doing so […]" (2006: 9). This hybridization has only truly kicked in within science very recently.

Mobile calculation and accountability

A very specific technology about to change contemporary scientific programming is the *Jupyter notebook,* a hybrid device between interactive computational environment and scientific documentation. Influenced by existing projects, such as Mathematica's notebook and IPython (Interactive Python), the notebooks were designed to "support the workflow of scientific computing, from interactive exploration to publishing a detailed record of computation" (Kluyver et al. 2016: 88). Considering that CLIMADA is delivered as Jupyter Notebook, we can use it to describe the functionalities of this technological device. The appearance of the notebooks is inconspicuous and discreet. They basically present themselves as simple HTML websites to be displayed in any web browser. This visual appearance is a stark understatement, as the whole fairly complex CLIMADA model is visible, operational and modifiable from within the notebook. A browser window is opened showing the file structure of the model when Jupyter and CLIMADA are initiated on a laptop.[68] One can browse within the folder and file structure, showing all elements of the CLIMADA programming code.

In our case, the data of these files are stored locally on a Macbook Pro laptop. However, the same setting would be equally deployable on a distributed cloud-computing environment,[69] thereby enabling more intensive operations computationally, such as machine learning. Operations within CLIMADA can be undertaken from within the Notebook

68 Via the Integrated Development Environment (IDE) Anaconda. https://www.anaconda.com/, retrieved on April 3, 2019.

69 E.g. an Amazon Elastic Compute Cloud (EC2). See https://docs.aws.amazon.com/dlami/latest/devguide/setup-jupyter.html, retrieved on July 5, 2019.

files within the folder structure, in our case '1_main_climada.ipynb,' shown in Figure 36. The element in focus is the calculation of 'exposures' within CLIMADA.

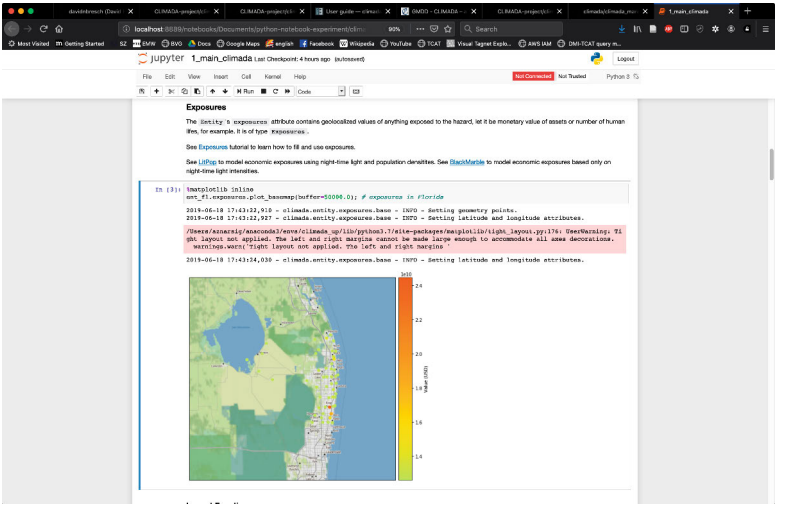


Figure 36: One of CLIMADAs Jupyter notebooks, the file '1_main_climada.ipynb.' Source: Own screenshot

The explanation of exposures says:

> **Exposures**
>
> The `Entity`'s `exposures` attribute contains geolocalized values of anything exposed to the hazard, let it be monetary value of assets or number of human lifes, for example. It is of type `Exposures`.

This general explanation is followed by a link to a tutorial (another notebook) that helps one to "learn how to fill and use exposures." The notebook then links to two specific submodels (or classes), 'LitPop' and 'BlackMarble,' which are alternative methods to model exposure to natural hazards.

See Exposures tutorial to learn how to fill and use exposures.

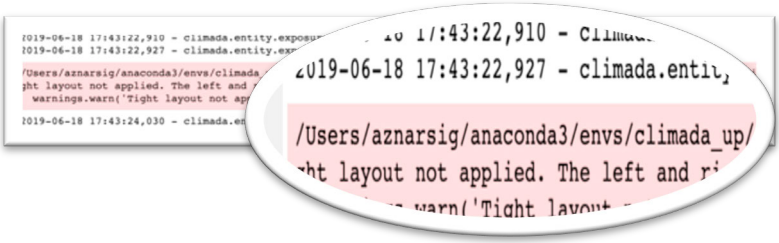
See LitPop to model economic exposures using night-time light and population densitites. See BlackMarble to model economic exposures based only on night-time light intensities.

The notebooks of LitPop and BlackMarble are both connected to the main CLIMADA notebook. Therefore, one can change elements within BlackMarble, which will be taken into account within any further CLIMADA calculation. It then follows a grayed-out text box, which highlights the calculative element of this notebook block:

```
%matplotlib inline
ent_fl.exposures.plot_basemap(buffer=50000.0); # exposures in Florida
```

This is a command line written in Python. It tells CLIMADA to plot exposures according to functions defined earlier in the text of the notebook. If one presses 'ALT + ENTER' on the keyboard, the notebook will calculate the exposures to a natural hazard in the territory of Florida and plot the results as dots in false colors on a map of the US state. The visual representation is enabled by calling up 'matplotlib,' a standard Python package for visualization purposes. In this particular case, the calculation takes less than one second.

The notebook then acts as a log file documenting the calculations performed by the machine (in this case, the CPU of my laptop):



The notebook does not only document calculations performed successfully but can return concrete error messages. In the present case, it informs us about a slight representational issue on the map: "Tight layout

not applied. The left and right margins cannot be made large enough to accommodate all axes decorations." In so doing, Jupyter also helps with troubleshooting in the interactive coding process.

The Jupyter notebook draws together many functionalities that are traditionally distributed among various technologies and artefacts, including:

- running code, such as a shell and command-line interface;
- organizing file structures and computational environments, such as an operating system;
- providing access to the entire programming code, as in a code repository;
- writing programming code, as in a code editor;
- producing and displaying laid out diagrams and formatted text, as in word processors and design software;
- documenting the scientific process, results and methodology, as in software user manuals and methods chapters of a publication;
- enabling data exploration and analysis, similar to proprietary software for visual analytics;
- providing a procedural tool for scientists to structure their work, similar to notes, post-its and various organization software;
- facilitating the collaboration in spatially distributed teams using cloud computing environments;
- facilitating replicability and reuse; and
- enabling various forms of monitoring and evaluation.

The Jupyter Notebook draws these functionalities together in one place, which can be easily accessed via a web browser. It provides an overview and possibilities for the manipulation and control of other distributed and fluid elements, such as data, programming code, computation, visualization and documentation. Within the daily programming practice, it is irrelevant whether these elements are stored on and retrieved from the physical computer located in front of the researcher or based on a distributed cloud computing infrastructure, such as the PIK's supercomputer, or Amazon's AWS.[70] As a matter of fact, commercial cloud computing services, such as AWS and Google Cloud, provide detailed instructions on how to set up Jupyter Notebooks within their infrastructures.[71]

Mobile modeling

I propose to subsume the discussed entanglement of practices and infrastructures under the term *mobile modeling*. Mobile modeling in science aims at profiting from the power of today's distributed computing technologies and infrastructures. Mobile modelers use the same programming languages and rely on similar software packages as programmers within the global knowledge economy. As a result, they are able to benefit from a powerful community of practice around data analysis ('data-science') from its continuously optimized software stacks and its powerful cloud computing infrastructures. The strategies of mobilization discussed above (pythonizing, packaging, wrapping code, tapping distributed communities) amplify the possibilities of scientific modeling endeavors. However, they also create considerable challenges for

70 Amazon Web Services.

71 For a detailed instruction in AWS, see https://docs.aws.amazon.com/dlami/latest/devguide/setup-jupyter.html, retrieved on April 3, 2019.

trust, control, performance and scientific soundness. Packages such as MatPlotLib are frequently updated to ensure the performance within a fluid environment of gradually evolving software stacks and infrastructural entanglements. Regarding scientific programming, this means that experiments have often been carried out with versions of software packages that might no longer be operational or available. Mobile modelers are trying to stabilize the increasingly distributed and fluid elements of their modeling environment with devices such as the Jupyter Notebook in order to keep some control over the scientific process. It is not surprising that the Jupyter notebook is presented by its developers as the Swiss army knife for future computational and data science. It appears like a technological fix to the perceived reproducibility crisis in science (Baker 2016; Kitchin 2014a; Marwick 2015), which is vividly debated in fields such as climate impact research.

Mobile modeling is essentially a forward-looking practice concerned about the expected mobility of modeling technology in the future, paralleled with strategies for prospective stabilization (also see the discussion in the next chapter). It always gives preferential treatment to technologies and infrastructures with capabilities of dealing with the distributiveness and fluidity of components.