HazImp Documentation

Release 1.0

Geoscience Australia

Mar 15, 2021

CONTENTS

HazImp is a tool for determining the impact due to natural hazards. It can be used to calculate damage to sites, given exposure and hazard information. It is command line based and can be executed in parallel.

CHAPTER

ONE

USER GUIDE

1.1 Introduction

HazImp is used to simulate the loss of value to structures from natural hazards using vulnerability curves. Generally the input information is hazard, such as a wind speed raster and exposure. The exposure information is currently supplied as a csv file, with structure locations given in latitude and longitude. This is combined with vulnerability curve information, described in an xml file. Figure 1.1 is an example of a vulnerability curve, showing a hazard value of the x-axis and the loss associated with that hazard on the y-axis:

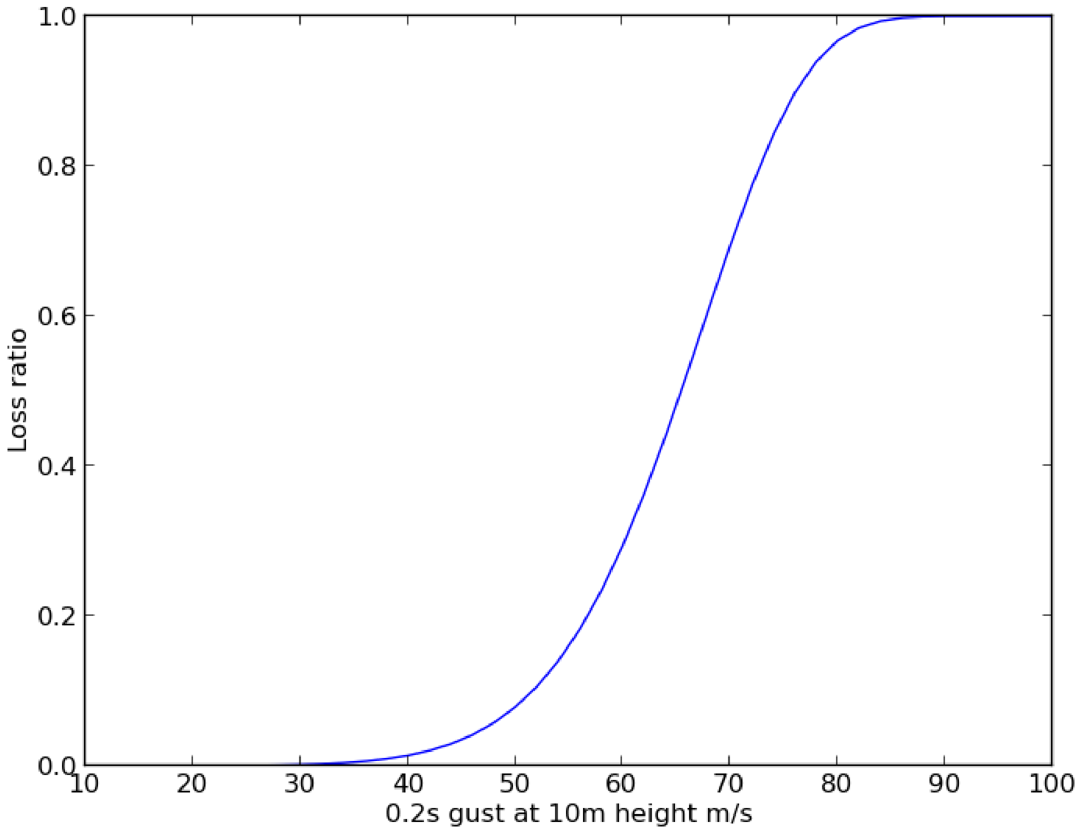

Fig. 1: *An example vulnerability curve.*

1.2 Quick how-to

Follow the install notes in the README.md file.

A configuration file can be used to define a HazImp simulation. The configuration file is described using yaml, a data serialisation format. HazImp can also be used by another Python application, by passing the configuration infomation in as a dictionary.

For example, to run a wind example do:

```
cd examples/wind
python ../../hazimp/main.py  -c wind_v5.yaml
```

The -c specifies the configuration file.

HazImp can also be ran in parallel, using mpirun. For example:

```
mpirun -np 4 python ../../hazimp/main.py  -c wind_v5.yaml
```

There are a suite of HazImp tests to test the install and code during software developemnt. To run these, in the root HazImp directory do;:

```
./all_tests
```

1.3 Templates

The simplest way to use HazImp is with a template. There is currently a wind template and a flood template. Templates take into account internal vulnerability curves and the data flow needed to produce loss information, simplifying the configuration file.

Note: The order of key/value pairs in the sample configuration files is important. The code will raise a *RuntimeError* if the order is incorrect.

1.4 Wind Template

Given gust information from TCRM and point exposure data the loss associated with each site is calculated using the wind template.

Here is the example wind configuration file (from examples/wind), which uses the wind template.

```
#  python hazimp.py -c wind_nc.yaml
 - template: wind_nc
 - vulnerability_filename: domestic_wind_vuln_curves.xml
 - vulnerability_set: domestic_wind_2012
 - load_exposure:
    file_name: WA_Wind_Exposure_2013_Test_only.csv
    exposure_latitude: LATITUDE
    exposure_longitude: LONGITUDE
 - load_wind:
    file_list: gust01.nc
    file_format: nc
```

(continues on next page)

(continued from previous page)

```
    variable: wndgust10m
  - calc_struct_loss:
    replacement_value_label: REPLACEMENT_VALUE
  - save: wind_impact.csv
  - aggregate:
    boundaries: SA1_2016_AUST.shp
    boundarycode: SA1_MAIN16
    impactcode: SA1_CODE
    save: gust01_impact.shp
```

The first line is a comment, so this is ignored. The rest of the file can be understood by the following key value pairs;

template The type of template to use. This example describes the *wind_nc* template.

load_exposure This loads the exposure data. It has 3 sub-sections:

> ***file_name*** The name of the csv exposure file to load. The first row of the csv file is the title row.
>
> ***exposure_latitude*** The title of the csv column with latitude values.
>
> ***exposure_longitude*** The title of the csv column with longitude values.

There are some pre-requisites for the exposure data. It must have a column called `WIND_VULNERABILITY_FUNCTION_ID` which describe the vulnerability functions to be used. It must also have a column called "WIND_VULNERABILITY_SET" which describes the vulnerability set to use (see below for more details).

load_wind This loads the hazard data. It can have up to three subsections;

> ***file_list*** A list of raster wind hazard files (one or more). The file format can be ascii grid, geotiff or netcdf (or potentially any raster format recognised by GDAL, but these are all that have ben tested to date).
>
> ***file_format*** This specifies the data format - specifically used for netcdf, where the string 'nc' should be used.
>
> ***variable_name*** For use when the file format is 'nc'. This specifies the name of the variable in the netcdf file that contains the hazard data.
>
> The values in the file must represent `0.2s gust at 10m height m/s`, since that is the axis of the HazImp wind vulnerability curves.

vulnerability_filename The path to a correctly formatted vulnerability curve file. This is an xml file produced using *hazimp_preprocessing/curve_data/create_vuln_xml.py*

vulnerability_set This defines the suite of vulnerability curves to use. A vulnerability file may contain a large number of different vulnerability functions that can be applied to the same exposure assets. This option defines which set to use from that vulnearbility file. The vulnerability set is used to calculate the `structural_loss_ratio` given the `0.2s gust at 10m height m/s`.

calc_struct_loss This will multiply the replacement value and the `structural_loss_ratio` to get the `structural_loss`.

> ***replacement_value_label*** The title of the exposure data column that has the replacement values.

save The file where the results will be saved. All the results to calculate the damage due to the wind hazard are saved to file. The above example saves to a csv file, since the file name ends in *.csv*. This has the disadvantage of averaging data from multiple wind hazards. The information can also be saved as numpy arrays. This can be done by using the *.npz* extension. This data can be accessed using Python scripts and is not averaged.

1.4.1 Using permutation to understand uncertainty in vulnerability

In many regions (in Australia), the attributes of individual buildings are unknown, but are recorded for some statistical area (e.g. suburb, local government area). In this case, the vulnerability curve assigned to a building may not be precisely determined, which can lead to uncertainty in the impact for a region.

To overcome this, users can run the impact calculation multiple times, while permuting the vulnerability curves for each region (suburb, local government area, etc.). This requires some additional entries in the template file.

exposure_permutation This describes the exposure attribute that will constrain the permutation, and the number of permuations.

groupby The field name in the exposure data by which the assets will be grouped.

iterations The number of iterations to perform

Example:

```
- exposure_permutation:
    groupby: MB_CODE
    iterations: 1000
```

1.4.2 Aggregation

aggregation This determines the way HazImp will aggregate results

groupby The exposure attribute that will be used to aggregate results. It is strongly recommended to use the same attribute as used for the exposure permutation.

kwargs A list of fields that will be aggregated to the level identified above. Each entry under this section must match an output field (`structural_loss_ratio`, `structural_loss`, `REPLACEMENT_VALUE`), followed by a Python-style list of statisticts to calculate: e.g. `mean`, `std` or `sum`.:

```
kwargs:
  structural_loss_ratio: [mean, std]
  structural_loss: [mean, sum]
  REPLACEMENT_VALUE: [mean, sum]
```

save_agg The file where the aggregated results will be saved.

This option has only been implemented in the `wind_nc` and `wind_v5` templates at this time (June 2020).

1.5 Flood Template - Structural Damage

The structural damage flood template is very similar to the the wind template. This is an example structural damage flood template;:

```
#  python ../../hazimp/hazimp.py -c list_flood_v2.yaml
# Don't have a scenario test automatically run this.
# Since the file location is not absolute,
- template: flood_fabric_v2
- floor_height_(m): .3
- load_exposure:
    file_name:  small_exposure.csv
    exposure_latitude: LATITUDE
    exposure_longitude: LONGITUDE
```

(continues on next page)

(continued from previous page)

```
- hazard_raster:  depth_small_synthetic.txt
- calc_struct_loss:
    replacement_value_label: REPLACEMENT_VALUE
- save: flood_impact.csv
```

The first 4 lines are comments, so they are ignored. The new key value pairs are;

floor_height_(m) This is used to calculate the water depth above ground floor; water depth(m) - floor height(m) = water depth above ground floor(m)

hazard_raster A list of ascii grid hazard files to load or a single file. The file format is grid ascii. The values in the file must be `water depth(m)`, since that is the axis of the vulnerability curves.

1.6 Without Templates

1.7 Vulnerability functions

See the **:ref:`Preparing vulnerability curves`_** section for guidance on preparing vulnerability functions for use in HazImp.

1.8 Provenance tracking

The provenance of information used in generating an impact analysis is tracked using the **Prov_** module.

Contributions to the code base should incorporate appropriate provenance statements to ensure consistency.

.._Prov: https://prov.readthedocs.io/en/latest/

CHAPTER

TWO

INSTALLING HAZIMP

2.1 Getting the code

Download the HazImp source from GitHub.

You can either download a zip file containing the HazImp code, or clone the repository (if you have *git* installed) as follows:

Using ssh:

```
git clone git@github.com:GeoscienceAustralia/hazimp.git
```

Using HTTPS:

```
git clone https://github.com/GeoscienceAustralia/hazimp
```

The source code includes a number of files that may help with installing the package dependencies.

2.2 Dependencies

HazImp uses Python and additional libraries.

There are several alternatives for suitable python environments.

2.2.1 conda

We recommend using conda to manage dependencies and run HazImp. Using conda helps to avoid version conflicts between different python packages and other dependencies. Download (wget) and install miniconda, then use the conda environment file *hazimp.yml* included with the software to install the set of compatible packages. conda is available for Linux, MacOS and Windows environments.

Once you have installed miniconda, create a new environment with a command such as:

```
conda create -f hazimp.yml
```

Before each session, remember to activate the corresponding environment, e.g. *conda activate hazimp*.

2.2.2 User install using system python

On some systems the existing system python installation may be made suitable.:

```
for x in pep8 coverage pyyaml pylint pandas prov pydot; do pip install --user $x ;␣
↪done
```

On MS-Windows:

```
for %x in (pep8, coverage, pyyaml, pylint, pandas, prov, pydot) do pip install
--user %x
```

NOTE:: This is not the exhaustive list of required packages. See the full list in *hazimp.yml*.

2.2.3 System python install

On Ubuntu systems, the following requires system administrator privileges.

```
sudo apt-get install python-numpy, python-scipy
sudo apt-get install python-gdal, python-yaml, python-coverage, pep8, pylint, pandas,␣
↪nose, prov, pydot
```

2.3 Install HazImp

Install HazImp into your python environment:

```
python setup.py install
```

Or, if you are interested in modifying HazImp, the following alternative install command will instead provide your python environment with links to the location where you have downloaded the HazImp source:

```
python setup.py develop
```

Please read the **`Contributing code`_** notes if you wish to modify HazImp.

To use HazImp, run *hazimp –help* from the command line. You can also verify the code using *./run_tests*.

2.4 Testing the installation

Users can test the installation with the **run_tests** script. This depends on the *nose* and *coverage* libraries for Python. The **run_tests** script is a shell script, so needs to be executed in a shell (e.g. *bash*, *sh* or *csh*).

On a Windows command line:

```
nosetests tests/ --with-doctest --cover-package=hazimp --with-xunit --xunit-
↪file=nosetests.xml --nocapture
```

CHAPTER

THREE

PREPARING VULNERABILITY CURVES

To calculate the level of impact from a hazard event, users need to provide information on the vulnerability of different building types to the hazard measure used. The vulnerability describes the level of damage, given a specific intensity of the hazard. The level of damage is given as a value between 0 and 1, and represents the ratio between cost of repair of a building, and the total replacement cost for that building. Typically, a value of 1 indicates the building will require complete replacement.

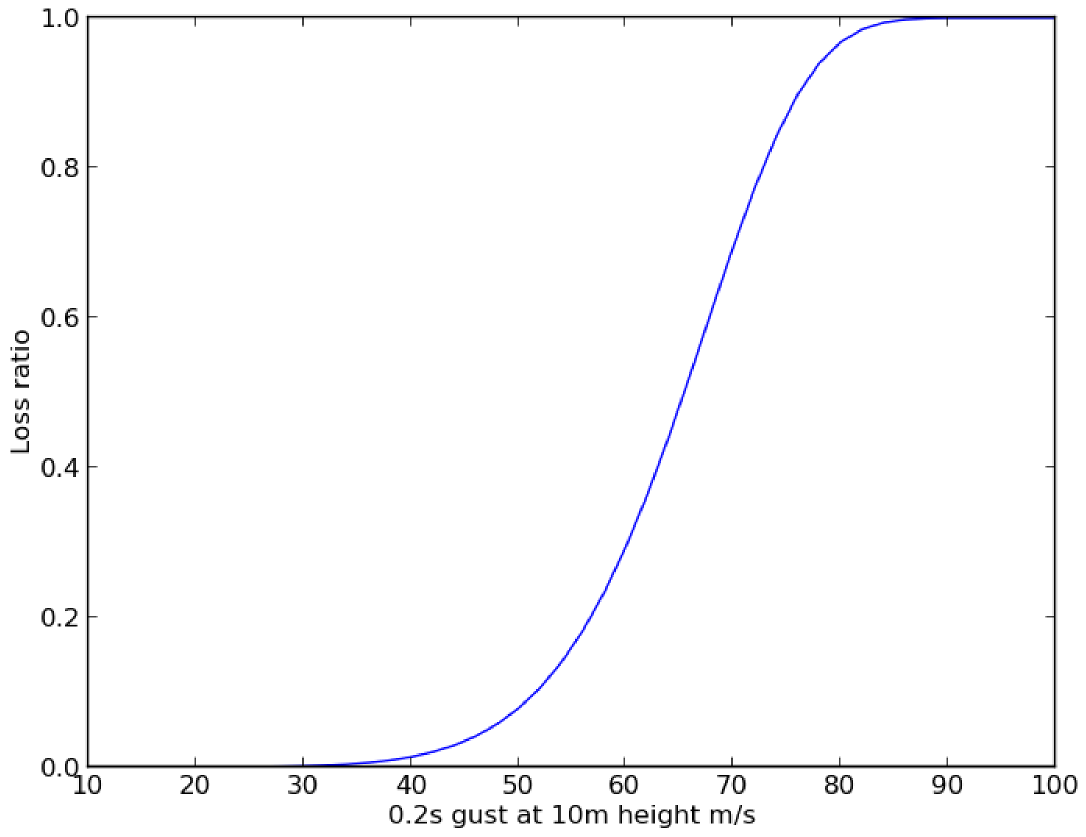

Fig. 1: *An example vulnerability curve.*

3.1 Format

The expected format of vulnerability functions is Natural hazard and Risk Markup language (NRML) 0.5, with the schema

```
<?xml version='1.0' encoding='utf-8'?>
<nrml xmlns="http://openquake.org/xmlns/nrml/0.5"
      xmlns:gml="http://www.opengis.net/gml">
    <vulnerabilityModel id="domestic_wind_2012" assetCategory="buildings"␣
↪lossCategory="structural">
            <description></description>
            <vulnerabilityFunction id="dw1" dist="LN">
                <imls imt="0.2s gust at 10m height m/s">17.0 20.0 22.0 24.0 26.0 28.0␣
↪30.0 32.0 34.0 36.0 38.0 40.0 42.0 44.0 46.0 48.0 50.0 52.0 54.0 56.0 58.0 60.0 62.
↪0 64.0 66.0 68.0 70.0 72.0 74.0 76.0 78.0 80.0 82.0 84.0 86.0 88.0 90.0 100.0</imls>
                <meanLRs>0 5.61E-05 0.000119676 0.000238983 0.000451491 0.000813598 0.
↪001407546 0.002349965 0.00380222 0.005982567 0.009180008 0.01376939 0.020226871 0.
↪029144114 0.041238619 0.05735623 0.078460253 0.10559985 0.13984902 0.182207241 0.
↪233455211 0.293965791 0.363483019 0.440901669 0.524104006 0.60993153 0.694373629 0.
↪773024088 0.841778089 0.897626918 0.93930181 0.967498609 0.98454129 0.99359009 0.
↪997731405 0.999330795 0.999839797 1</meanLRs>
                <covLRs>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0␣
↪0 0 0 0 0 0 0</covLRs>
            </vulnerabilityFunction>
    </vulnerabilityModel>
</nrml>
```

Vulnerability files are validated against the NRML 0.5 schema definition. If the file is not a vaild NRML file, an error will be raised. Note this validation does not do a complete check of the logical consistency of the values in the file. For example, NRML includes *meanLRs* and *covLRs*. The value of *covLRs* must be zero where the *meanLRs* values are zero. These internal consistency checks are not done by the validation check.

3.2 Structuring vulnerability functions

The folder *hazimp_preprocessing/curve_data* contains example formatted CSV files of vulnerability, which can be converted to the required XML format using the *create_vuln_xml.py* script.

The first row of the CSV file contains column labels, some of which are unique to each hazard. Key fields in the CSV include:

```
- `vulnerabilityFunctionID`
- `vulnerabilitySetID`
- `lossCategory`
- `defaultLoss`
- `IMT` - intensity measure type (the way the hazard is measured)
- `IML` - intensity measure levels (the hazard levels for which damage/loss ratios␣
↪are provided)
```

Additional fields may be included to facilitate demarcation of different vulnerability groups. For example, the `domestic_wind_vul_curves.csv` includes fields of `Region`, `Age`, `AS4055_class`, `Roof material` and `Wall material`. These attributes are all used to assign an appropriate vulnerability curve to the builiding. However, they are not used in the final XML dataset.

The `vulnerabilityFunctionID` value is a unique identifier to each vulnerability function. These are used to assign a curve to each asset contained in the input exposure data, and that column must be labelled

WIND_VULNERABILITY_FUNCTION_ID.

3.3 Generating a vulnerability xml file

Once an appropriate CSV file has been generated, the curve data is converted to an XML file using the `create_vuln_xml.py` script. Edit the lines below `if __name__ == "__main__":` to turn on the processing of individual CSV files. For example, to process a CSV file called `domestic_wind_vuln_curves.csv`, there should be a line like the following:

```
if True:
    csv_curve2nrml('domestic_wind_vul_curves.csv',
                   'domestic_wind_vul_curves.xml')
```

Then run the script at the command line:

```
cd hazimp_preprocessing/curve_data
python create_vuln_xml.py
```

This should generate a file called `domestic_wind_vul_curves.xml` in the working directory (`hazimp_preprocessing/curve_data`). This xml file can then be referenced in the *Templates* to run HazImp.

CHAPTER

FOUR

TEMPLATES

The simplest way to use HazImp is with a template, which sets up a `PipeLine` to run a collection of `Job` functions. There is currently a wind template and a flood template. Templates take into account internal vulnerability curves and the data flow needed to produce loss information, simplifying the configuration file.

Templates are used to set out the flow of processing invoked in separate `Job` functions that are then connected into a `PipeLine` that is subsequently executed.

Note: Because some of the `Job` functions in the templates are essential, the order of key/value pairs in the configuration files is important. The code will raise a *RuntimeError* if the order is incorrect, or if a mandatory configuration key is missing. This is even more important if not using the pre-defined templates.

We take the example of the *wind_v4* template. It sets up the following job sequence in a specific order:

```
#. LOADCVSEXPOSURE - load the exposure data
#. LOADRASTER - load the wind raster data
#. LOADXMLVULNERABILITY - load the vulnerability functions
#. SIMPLELINKER - select a group of vulnerability functions - some vulnerability␣
↪files may have multiple sets of curves identified by `vulnerabilitySetID`
#. SELECTVULNFUNCTION - link the selected vulnerability function set (specified by␣
↪the `vulnerabilitySetID` option) to each exposure asset
#. LOOKUP - do a table lookup to determine the damage index for each asset, based on␣
↪the intensity measure level (i.e. the wind speed)
#. CALCSTRUCTLOSS - combine the calculated damage index with the building value to␣
↪calculate $$$ loss
#. SAVE - should speak for itself - saves the building level loss data
#. SAVEPROVENANCE - saves provenance data (like the version of HazImp, source of the␣
↪hazard data, etc.)
```

CHAPTER

FIVE

SAVING TO GEOSPATIAL FORMATS

Data can optionally be saved to a geospatial format that aggregates the impact data to spatial regions (for example suburbs, post codes).

aggregate This will activate the option to save to a geospatial format.

boundaries The path to a geospatial file that contains polygons to aggregate by

filename The path to the output geospatial file. This can be either an ESRI shape file (extension *shp*), a GeoJSON file (*json*) or a GeoPackage (*gpkg*). If an ESRI shape file is specified, the attribute names are modified to ensure they are not truncated/

impactcode The attribute in the exposure file that contains a unique code for each geographic region to aggregate by.

boundarycode The attribute in the *boundaries* file that contains the same unique code for each geographic region. Preferably the *impactcode* and *boundarycode* will be of the same type (e.g. *int* or *str*)

categories If True, and the *Including categorisations* job is included in the pipeline, the number of assets in each damage state defined by the *Including categorisations*, in each geographic region, will be added as an attribute to the output file. **Note the different spelling used in this section.**

Presently, HazImp will aggregate the following fields:

```
'structural_loss_ratio': 'mean',
```

Example:

```
- aggregate:
    boundaries: QLD_Mesh_Block_2016.shp
    file_name: QLD_MeshblockImpacts.shp
    impactcode: MB_CODE
    boundarycode: MB_CODE16
    categories: True
```

This option has only been implemented in the `wind_nc` and `wind_v5` templates at this time (June 2020).

CHAPTER

SIX

INCLUDING CATEGORISATIONS

Users may want to convert from numerical values to categorical values for output fields. For example, converting the structural loss ratio from a value between 0 and 1 into categories which are descriptive (see the table below).

Lower value	Uppper value	Category
0	0.02	Negligible
0.02	0.1	Slight
0.1	0.2	Moderate
0.2	0.5	Extensive
0.5	1.0	Complete

The *Categorise* job enables users to automatically perform this categorisation. Add the "categorise" section to the configuration file to run this job. This is based on the *pandas.cut* method.

categorise This will enable the *Categorise* job. The job requires the following options to be set.

field_name The name of the created (categorical) field in the *context.exposure_att DataFrame*

bins Monotonically increasing array of bin edges, including the rightmost edge, allowing for non-uniform bin widths. There must be (number of labels) + 1 values, and range from 0.0 to 1.0.

labels Specifies the labels for the returned bins. Must be the same length as the resulting bins.

6.1 Example

A basic example, which would result in the categories listed in the table above being added to the column "Damage state":

```
- categorise:
   field_name: 'Damage state'
   bins: [0.0, 0.02, 0.1, 0.2, 0.5, 1.0]
   labels: ['Negligible', 'Slight', 'Moderate', 'Extensive', 'Complete']
```

Another example with three categories. See the example configuration in *examples/wind/three_category_example.yaml*

```
- categorise:
   field_name: 'Damage state'
   bins: [0.0, 0.1, 0.5, 1.0]
   labels: ['Minor', 'Moderate', 'Major']
```

CHAPTER

SEVEN

AUTO GENERATED DOCUMENTATION

7.1 Pipeline

The purpose of this module is to provide objects to process a series of `Job` functions in a sequential order. The order is determined by the queue of `Job`.

Typically, a `PipeLine` is created through pre-defined *Templates* that have been built into HazImp already. These templates ensure the correct set of jobs are executed, in the correct order, for given use cases.

It's also possible to build a `PipeLine` manually, using the **:method:`add_job`** method to add more jobs.

Warning: The order of jobs in a `PipeLine` is important. The existing templates ensure correct order of the jobs. If creating a `PipeLine` manually, the user will be responsible for ensuring the correct order of jobs.

Note: Currently we include the `SaveProvenance` job in the templates, so a manually defined `PipeLine` will have to explicitly include that at the end to ensure provenance information is captured.

class hazimp.pipeline.**PipeLine**(*jobs_list=None*)

PipeLine allows to create a queue of jobs and execute them in order.

add_job(*a_job*)

Append a new job the to queue

run(*context*)

Run all the jobs in queue, where each job take input data and write the results of calculation in context.

Parameters **context** – Context object holding the i/o data for the pipelines.

7.2 Context

7.3 Templates

7.4 Aggregate

7.5 Raster

7.6 Miscellaneous

CHAPTER

EIGHT

CONTRIBUTING CODE

The preferred way to contribute to HazImp is to fork the main repository on GitHub:

1. Fork the project repository: click on the 'Fork' button near the top of the page. This creates a copy of the code under your account on the GitHub server.
2. Clone this copy to your local disk:

   ```
   $ git clone git@github.com:YourLogin/hazimp.git
   $ cd hazimp
   ```

3. Create a branch to hold your changes:

   ```
   $ git checkout -b my-feature
   ```

   and start making changes. Never work in the `master` branch!
4. Work on this copy on your computer using Git to do the version control. When you're done editing, do:

   ```
   $ git add modified_files
   $ git commit
   ```

   to record your changes in Git, then push them to GitHub with:

   ```
   $ git push -u origin my-feature
   ```

Finally, go to the web page of the your fork of the hazimp repo, and click 'Pull request' to send your changes to the maintainers for review. This will send an email to the committers. **Pull requests should only be made against the 'develop' branch of the repository.** Pull requests against the 'master' branch will be refused and requested to resubmit against the 'develop' branch.

A demonstration of the git workflow used for HazImp can be seen at https://leanpub.com/git-flow/read.

(If any of the above seems like magic to you, then look up the Git documentation on the web.)

It is recommended to check that your contribution complies with the following rules before submitting a pull request:

- All public methods should have informative docstrings with sample usage presented as doctests when appropriate.
- When adding additional functionality, provide at least one example script in the `examples/` folder. Have a look at other examples for reference. Examples should demonstrate why the new functionality is useful in practice.
- At least one paragraph of narrative documentation with links to references in the literature (with PDF links when possible) and an example.
- Include clear provenance statements in any new functionality that generates output, using the 'prov' library.

You can also check for common programming errors with the following tools:

- Code with good unittest coverage, check with:

  ```
  $ pip install nose coverage
  $ nosetests --with-coverage path/to/tests_for_package
  ```

- No pyflakes warnings, check with:

  ```
  $ pip install pyflakes
  $ pyflakes path/to/module.py
  ```

- No PEP8 warnings, check with:

  ```
  $ pip install pep8
  $ pep8 path/to/module.py
  ```

- AutoPEP8 can help you fix some of the easy redundant errors:

  ```
  $ pip install autopep8
  $ autopep8 path/to/pep8.py
  ```

8.1 Issues

A great way to start contributing to HazImp is to pick an item from the list of Issues in the issue tracker. (Well there are none there yet, but we will be putting some up soon!) Resolving these issues allow you to start contributing to the project without much prior knowledge. Your assistance in this area will be greatly appreciated by the more experienced developers as it helps free up their time to concentrate on other issues.

8.2 Documentation

We are in the process of creating sphinx based documentation for HazImp. Any help in setting this up will be gratefully accepted!

At present you will find the user guide in the docs folder.

We are glad to accept any sort of documentation: function docstrings, reStructuredText documents (like this one), tutorials, etc. reStructuredText documents live in the source code repository under the docs/ directory.

When you are writing documentation, it is important to keep a good compromise between mathematical and algorithmic details, and give intuition to the reader on what the algorithm does. It is best to always start with a small paragraph with a hand-waving explanation of what the method does to the data and a figure (coming from an example) illustrating it.

CHAPTER

NINE

HISTORY

9.1 1.0.0 (2021-03-12)

- Add provenance records
- Add categorisation, tabulation and aggregation
- Implemented AWS S3 download and upload funcitonality. Also allow the configuration file to be stored on S3 as well.
- Update hazimp-tests.yml - add coveralls, remove flake8 and pytest install
- Calculate percentages of assets in damage state tabulation
- Updates to documentation (user guide and in-code)
- Add NRML schema from https://github.com/gem/oq-nrmllib/tree/master/openquake/nrmllib/schema
- Validate xml-format vulnerability files prior to execution
- Add support for configuring choropleth aggregation fields via the configuration file
- Add aggregation functions to provenance statement
- Add support for generic 'hazard_raster' to wind templates
- Switched template instantiation to use config dictionary, removed use of config list
- Update find_attributes to use config dictionary for locating attributes, and support lookup via a ordered list of keys to ease deprecation
- add domestic eq vulnerability curves in MMI (#35)
- Increase unit test coverage (#36)
- Update to NRML 0.5 schema
- Add EQ template and example
- Split templates into new module separated by hazard type
- Enable hazard_raster attribute_label to be configured
- Improve memory usage for large raster input: No longer reads the entire raster into memory, reads only the cells defined in the exposure data, removed the ability to pass an in-memory array via 'from_array' and added a 'ThreadPoolExecutor' for some performance improvement when reading hazard data for large exposure datasets
- Change default aggregate output format to GeoJSON (#42)

9.2 0.6 (2020-08-13)

- WIP: First steps on provenance - Initially only implemented in the wind_nc and wind_v5 templates, but several of the jobs include provenance statements
- Add provenance for aggregation file
- Add categorisation and tabulation (pivot table)
- Adding some basic documentation for new functions
- Add updated v5 template
- Template file for tcimpact processing
- WIP: PEP8 conformance
- TCRM-47: Implemented S3 download and upload functionality.
- TCRM-47: Allow config file to be from S3 as well.
- TCRM-88: Moving logger.INFO output to stdout from stderr. Fixed compiler warning.
- Documentation updates
- TCRM-90: Reduce memory usage which require 2 times memory for source and destination arrays. (#18)
- Set raster dtype to GDT_Float32: May not necessarily always be correct, however the previous default was integer which worked for the test cases, except when replacing missing values with *numpy.NaN* (which is ostensibly a float value).
- Fix unittest suite by mocking prov module: Patching various methods under *prov.model.ProvDocument* to allow the test suite to run without error. The tests set up a dummyy context, but missed the provenance module, so we're using *mock* to patch the methods that are called in some tests.

9.3 0.5 (2020-07-09)

- Implement S3 access

9.4 0.4 (2020-06-29)

- Provenance tracking and aggregation added

PYTHON MODULE INDEX

INDEX