"OpenQuake: Calculate, share, explore"

Risk Modeller's Toolkit - User Guide

PUBLISHED BY GEM FOUNDATION

GLOBALQUAKEMODEL.ORG/OPENQUAKE

Citation

Please cite this document as:
Casotto, C. (2015) OpenQuake Risk Modeller's Toolkit - User Guide. *Global Earthquake Model (GEM). Technical Report 2015-06.*
doi: 10.13117/GEM.OPENQUAKE.MAN.RMTK.1.0/01, 60 pages.

Disclaimer

License

First printing, June 2015

Contents

Preface

To be completed by Vitor.

The goal of this book is to provide a comprehensive and transparent description of the methodologies adopted and implemented in the Risk Modeller's Toolkit (RMTK).

1. Introduction

1.1 Getting started

The Risk Modeller's Toolkit makes extensive use of the Python programming language and the web-browser based interactive IPython notebook interface. As with OpenQuake, the preferred working environment is Ubuntu (12.04 or later) or Mac OS X. At present, the user must install the dependencies manually. An effort has been made to keep the number of additional dependencies to a minimum. More information regarding the current dependencies of the toolkit can be found at http://github.com/GEMScienceTools/rmtk.

The current dependencies are:

- Numpy and Scipy (included in the standard OpenQuake installation)
- Matplotlib (http://matplotlib.org/)

The Matplotlib library can be installed easily from the command line by:

```
~$ sudo pip install matplotlib
```

To enable usage of the rmtk within any location in the operating system, OS X and Linux users should add the path to the rmtk folder to their profile file. This can be done as follows:

1. Using a command line text editor (e.g. VIM or Emacs), open the `~/.profile` folder as follows:

   ```
   ~$ vim ~/.profile
   ```

2. At the bottom of the profile file (if one does not exist it will be created) add the line:

   ```
   export PYTHONPATH=/path/to/rmtk/folder/:$PYTHONPATH
   ```

   Where `/path/to/rmtk/folder/` is the system path to the location of the rmtk folder (use the command `pwd` from within the rmtk folder to view the full system path).
3. Reload the profile file using the command

   ```
   ~$ source ~/.profile
   ```

At present, the recommended approach for Windows users is to run Ubuntu Linux 12.04 within a Virtual Machine and install the rmtk following the instructions above. Up-to-date VirtualBox images containing the OpenQuake-engine and platform, and the Hazard and Risk Modeller's Toolkits are available here: http://www.globalquakemodel.org/openquake/start/download/

1.2 Current features

To be completed by Anirudh.

The Risk Modeller's Toolkit is currently divided into two sections:

1. These functions are intended to address the modeller's needs for defining vulnerability curves, implementing methodologies differing for level of complexity and for the input data available for the buildings under study. GEM analytical vulnerability guidelines have been integrated in this tool and some of the methodologies indicated have been already implemented in the library.
2. These functions are intended to address the needs of visualising the results of the calculations performed with the OpenQuake-engine.

1.3 Organization

This manual is designed to explain the various functions in the toolkit, to provide the theoretical background behind them, and to guide the modeller in the use of the rmtk within the "IPython Notebook" environment. This novel tool implements Python inside a web-browser environment, permitting the user to execute real Python workflows, whilst allowing for images and text to be embedded. Its use is encouraged especially for beginner Python users for a more visual application of the rmtk.

The IPython Notebook comes installed from version 1.0 of IPython, that can be installed from the Python package repository by entering:

```
~$ sudo pip install ipython
```

A notebook session can be started via the command:

```
~$ ipython notebook --pylab inline
```

The tutorial itself does not specifically require a working knowledge of Python. However, an understanding of the basic Python data types is highly desirable. Users who are new to Python are recommended to familiarise themselves with Appendix A of this tutorial.

The *rmtk* is currently subdivided into two classes of tools, the Vulnerability and Plotting tools, presented in Chapter 2 and Chapter 3 of this tutorial respectively. In the Vulnerability chapter the vulnerability methodologies implemented are classified in Non-linear Static (NLS) and Non-linear Dynamic (NLD) according to the structural analysis type performed to assess the response of the building. These two main sections (NLS and NLD) are organised as follows:

- General Introduction.
- Getting Started, where it is explained what files need to be executed to start the vulnerability analysis, and what options are available to call the preferred methodology and to input the preferred data type.
- Description of the methodologies.

Within the description of each methodology the user can find the following subsections:

- Theoretical description of the method.
- Description and examples of the inputs.
- Description of the workflow.

A summary of the algorithms available in the present version is given in Table 1.1.

Feature	Algorithm
Non-linear Static	Cr-based (Ruiz-Garcia and Miranda, 2007) Spo2ida (Vamvatsikos and Cornell, 2006) R-$/mu$-T-based (Dolsek and Fajfar, 2004)
Non-linear Dynamic	DPM-based (Silva et al. 2013) Ida-postprocessing (Vamvatsikos and Cornell, 2002)

Table 1.1 – *Current algorithms in the RMTK*

2. Plotting

The OpenQuake-engine is capable of generating several seismic hazard and risk outputs, such as loss exceedance curves, seismic hazard curves, loss and hazard maps, damage statistics, amongst others. Most of these outputs are stored using the Natural Risk Markup Language (NRML), or simple comma separated value (CSV) files. The Plotting module of the Risk Modeller's Toolkit allows users to visualize the majority of the OpenQuake-engine results, as well as to convert them into other formats compatible with GIS software (e.g. QGIS). Despite the default styling of the maps and curves defined within the Risk Modeller's Toolkit, it is important to state that any user can adjust the features of each output by modifying the original scripts.

2.1 Plotting damage distribution

Using the Scenario Damage Calculator (**SilvaEtAl2014a**) of the OpenQuake-engine, it is possible to assess the distribution of damage for a collection of assets considering a single seismic event. These results are comprised of damage per building typology, total damage distribution, and distribution of collapses in the region of interest.

2.1.1 Plotting damage distribution

This feature of the Plotting module allows users to plot the distribution of damage across the various vulnerability classes, as well as to the total damage distribution. For what concerns the former result, it is necessary to set the path to the output file using the parameter `tax_dmg_dist_file`. It is also possible to specify which vulnerability classes should be considered, using the parameter `taxonomy_list`. However, if a user wishes to consider all of the vulberability classes, then this parameter should be left empty. It is also possible to specify is a 3D plot containing all of the vulnerability classes should be generated, or a instead a 2D plot per vulnerability class. For follow the former option, the parameter `plot_3d` should be set to `True`. It is important to understand that this option leads to a plot of damage fractions for each vulnerability class, instead of the number of assets in each damage state. An example of this output is illustrated in Figure 2.1.

In order to plot the total damage distribution (considering the entire colection of assets), it is necessary to use the parameter `total_dmg_dist_file` to define the path to the respective output file.Figure 2.2 presents an example of this type of output.

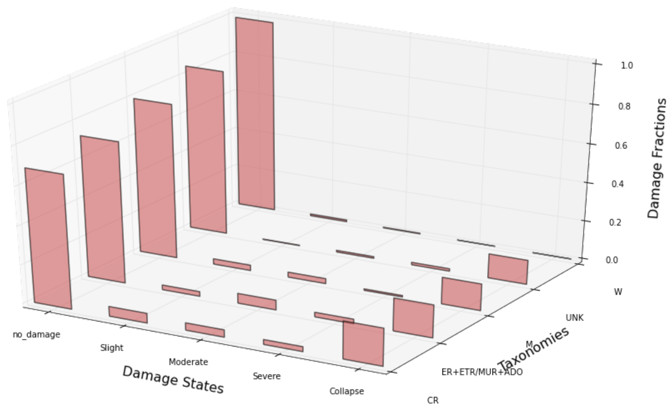

Figure 2.1 – *Damage disaggregation per vulnerability class.*

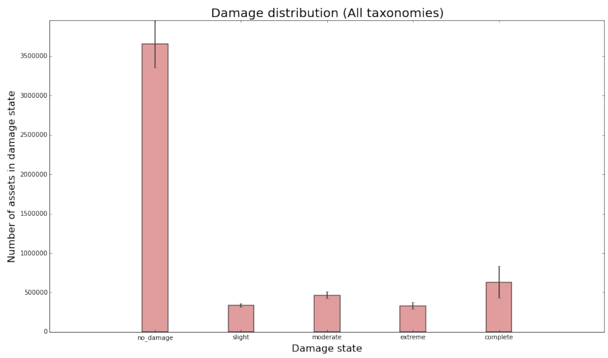

Figure 2.2 – *Total damage distribution.*

2.1.2 Plotting collapse maps

The OpenQuake-engine also generates an output defining the spatial distribution of the mean (and associated standard deviation) of assets in the last damage state (usually representing collapse or complete damage). The location of this output needs to be specified using the parameter `collapse_map`. Then, it is necessary to specify whether the user desires a map with the aggregated number of collapses (i.e. at each location, the mean number of collapses across all of the vulnerability classes are summed) or a map for each vulnerability class. Thus, the following options are permitted:

1. Aggregated collapse map only.
2. Collapse maps per taxonomy only.
3. Both aggregated and taxonomy-based.

The plotting option should be specified using the parameter `plotting_type`, and the location of the exposure model used to perform the calculations must be defined using the variable `exposure_model`. A number of other parameters can also be adjusted to modify the style of the resulting collapse map as follows:

- `bounding_box`: If set to 0, the Plotting module will calculate the geographical distribution of the assets, and adjust the limits of the map accordingly. Alternatively, a user can also specify the minimum/maximum latitude and longitude that should be used i nthe creation of the map.

- `marker_size`: This attribute can be used to adjust the size of the markers in the map.
- `log_scale`: If set to `True`, it will apply a logarithmic scale on the color scheme of the map, pottentially allowing a better visualization of the variation of the numbers of collapses in the region of interest.

An example of a collapse map is presented in Figure 2.3.

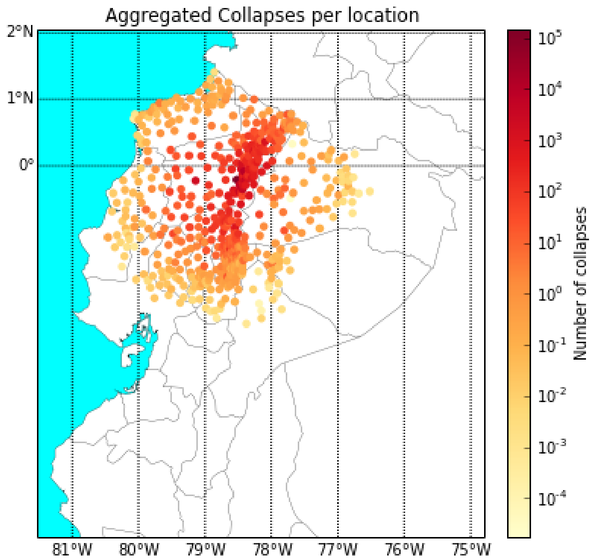

Figure 2.3 – *Spatial distribution of the mean number of collapses.*

2.2 Plotting hazard and loss curves

Using the Classical PSHA-based or Probabilistic Event-based Calculators (**SilvaEtAl2014a PaganiEtAl2014a**) of the OpenQuake-engine, it is possible to calculate seismic hazard curves for a number of locations, or loss exceedance curves considering a collection patially distributted assets.

2.2.1 Plotting hazard curves and uniform hazard spectra

A seismic hazard curve defines the probability of exceeding a number of intensity measure levels (e.g. peak ground acceleration or spectral acceleration) for a given interval of time (e.g. 50 years). In order to plot these curves, it is necessary to define the path to the output file in the parameter `hazard_curve_file`. Then, since each output file might contained a great number of haard curves, it is necessary to establish the location for each the hazard curve will be extracted. To visualize the list of locations comprised in the output file, the function `hazard_curves.loc_list` can be employed. Then, the choosen location must be provided to the plotting function (e.g. `hazard_curves.plot("81.213823|29.761172")`). An example of a seismic hazard curve is provided in Figure 2.4.

To plot uniform hazard spectra (UHS), a similar approach should be followed. The output file containing the uniform hazard spectra should be defined using the parameter `uhs_file`, and then a location must be provided to the plotting function (e.g.`uhs.plot("81.213823|29.761172")`). An example of uniform hazard spectra is illustrated in Figure 2.5.

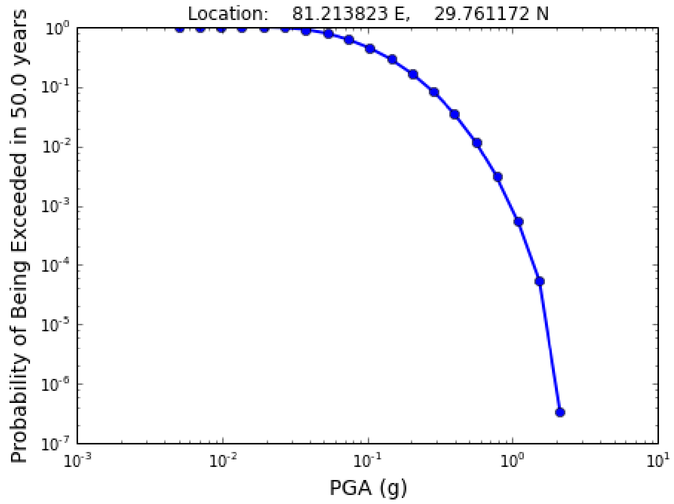

Figure 2.4 – *Seismic hazard curve for peak ground acceleration (PGA).*

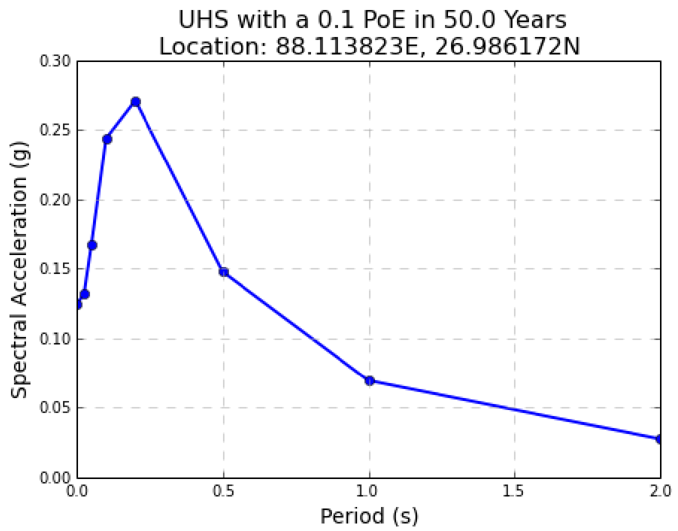

Figure 2.5 – *Uniform Hazard Spectra for a probability of exceedance of 10% in 50 years.*

2.2.2 Plotting loss curves

A loss exceedance curve defines the relation between a set of loss levels and the corresponding probability of exceedance within a given time span (e.g. a year). In order to plot these curves, it is necessary to define the location of the output file using the parameter `loss_curves_file`. Since each output file may contains a large number of loss exceedance curves, it is necessary to define for which assets will the loss curves be extracted. The parameter `assets_list` should be employed to define all of the chosen asset ids. These ids can be visualize directly on the loss curve output file, or on the exposure model used for the risk calculations. It is also possible to define a logarithimic scale for the x and y axis using the parameters `log_scale_x` and `log_scale_y`. A loss exceedance curve for a single asset is depicted in Figure 2.6.

2.3 Plotting hazard and loss maps

The OpenQuake-engine offers the possibility of calculating seismic hazard and loss (or risk) maps. To do so, it utilizes the seismic hazard or loss exceedances curves, to estimate the

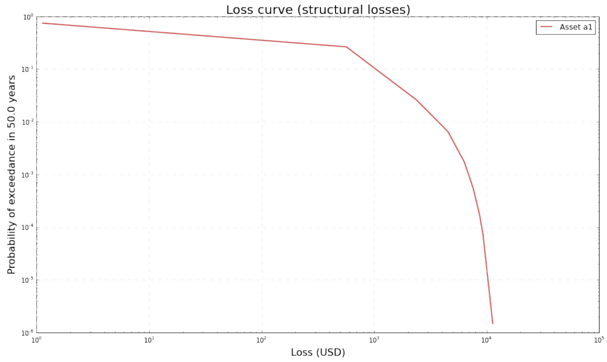

Figure 2.6 – *Loss exceedance curve.*

corresponding hazard or loss for the pre-defined return period (or probability of exceedance within a given interval of time).

2.3.1 Plotting hazard maps

A seismic hazard map provides the expected ground motion (e.g. peak ground acceleration or spectral acceleration) at each location, for a certain return period (or probability of exceedance within a given interval of time). To plot this type of maps, it is necessary to specify the location of the output file using the parameter `hazard_map_file`. An example hazard map is displayed in Figure 2.4.

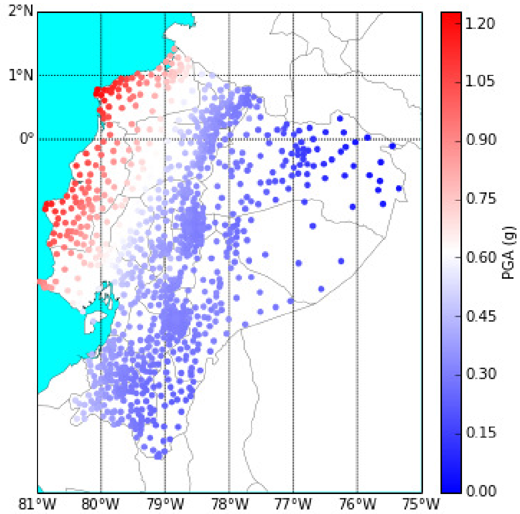

Figure 2.7 – *Seismic hazard map for a probability of exceedance of 10% in 50 years.*

2.3.2 Plotting loss maps

A loss map provides the estimated losses for a collection of assets, for a certain return period (or probability of exceedance within a given interval of time). It is important to understand that these maps are not providing the distribution of losses for a seismic event for the chosen return period, nor the losses whose sum would correspond to the aggreagted loss for the same return period. This type of maps is simply providing the expected loss for a specified level of frequency for each asset. To use this feature, it is necessary to define the parth ot the output file using the parameter `loss_map_file`, as well as the exposure model used to perform the risk calculations through the parameter `exposure_model`. Then, similarly to what was explained in section 2.1.2 for collapse maps, it is possible to follow three approaches to generate the loss maps:

1. Aggregated loss map only.
2. Loss maps per taxonomy only.
3. Both aggregated and taxonomy-based.

Then, there are a number of options that can be used to modify the style of the maps. These include the size of the marker of the map (`marker_size`), the geographical limits of the map (`bounding_box`), and the employment of a logarithmic spacing for the color scheme (`log_scale`). An example loss map for a single vulnerability class is presented in Figure 2.8.

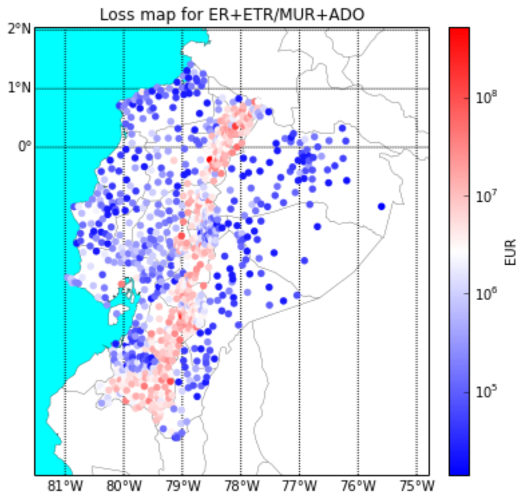

Figure 2.8 – *Loss (economic) map for a probability of exceedance of 10% in 50 years.*

As mentioned on the introductory section, it is also possible to convert any of the maps into a format (csv) easily readable by GIS software. To do so, it is necessary to set the parameter `export_map_to_csv` to `True`. As an example, a map containing the average annual losses for Ecuador has been converted to the `csv` format, and introduced into the QGIS software to produce the map presented in Figure 2.9.

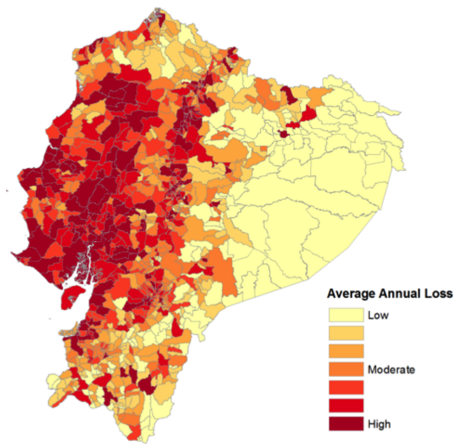

Figure 2.9 – *Average annual (economic) losses for Ecuador.*

3. Risk

The OpenQuake-engine currently generates the most common seismic hazard and risk results (e.g. hazard maps, loss curves, average annual losses). However, it is recognized that there are a number of other metrics that might not be of interest of the general GEM community, but fundamental for specific users. This module of the Risk Modeller's Toolkit aims to provide users with additional risk results and funtionalities.

3.1 Deriving Probable Maximum Losses (PML)

The Probabilistic Event-based Risk calculator (**SilvaEtAl2014a**) of the OpenQuake-engine is capable of calculating event loss tables, which contain a list of earthquake ruptures and associarted losses. These losses my refer to specific assets, or the sum of the losses from the entire building portfolio (aggregated loss curves). Using this module, it is possible to derive a probable maximum loss (PML) curves (i.e. relation between a set of loss levels and corresponding return peridos of exceedance), as illustrated in Figure 3.1.

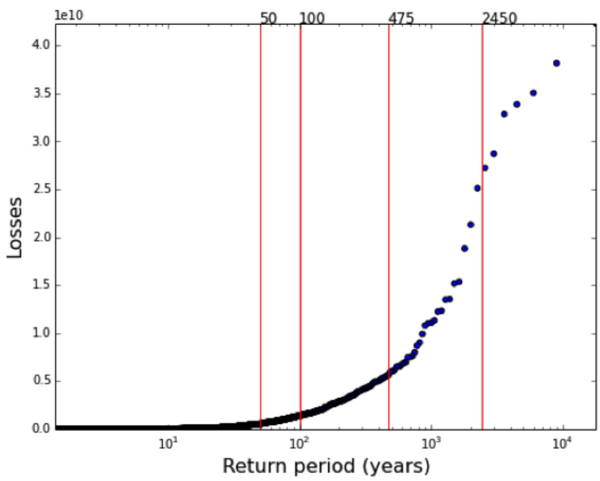

Figure 3.1 – *Probable Maximum Loss (PML) curve.*

To use this feature, it is necessary to use the parameter `event_loss_table_folder` to specify the location of the folder that contains the set of event loss tables and stochastic event sets. Then, it is also necessary to provide the total economic value of the building portfolio (using the variable `total_cost`) and the list of return periods of interest (using the variable `return_periods`). This module also offers the possibility of saving all of the information in `csv` files, which can be used in other software (e.g. Microsoft Excel) for other purposes. To do so, the parameters `save_elt_csv` and `save_ses_csv` should be set to `True`.

3.2 Selecting a logic tree branch

When a non-trivial logic-tree is used to capture the model uncertainty in the source model or in the choice of an appropriate ground motion prediction equation (GMPE) for each of the tectonic region types in teh region considered, OpenQuake can calculate the hazard curves for each end-branch of the logic-tree individualy. Now, if a risk modeller wishes to estimate damage or losses using one or a few of these branches only, it is useful to compare the hazard curves for the chosen branch with the mean hazard curve. Depending upon the distance of the hazard curve for a particular branch from the mean hazard curve, the risk modeller may wish to choose the branches for which the hazard curves are closest to the mean hazard curve. This Python script and corresponding IPython notebook allow the risk modeller to list the end-branches for the hazard calculation, sorted in increasing order of the distance of the branch hazard curve from the mean hazard curve. Currently, the distance metric used for performing the sorting is the root mean square distance.

4. Vulnerability

4.1 Introduction

To be completed by Anirudh.

4.2 Definition input models

To be completed by Anirudh.

4.2.1 Definition of capacity curves

To be completed by Anirudh.

4.2.2 Definition of ground motion records

To be completed by Anirudh.

4.2.3 Definition of damage model criterium

To be completed by Anirudh.

4.3 Model generator

To be completed by Vitor.

4.3.1 Generation of capacity curves using DBELA

To be completed by Vitor.

4.3.2 Generation of capacity curves using SP-BELA

To be completed by Vitor.

4.3.3 Generation of capacity curves using point dispersion

To be completed by Vitor.

4.4 Conversion from MDOF to SDOF

To be completed by Vitor.

4.4.1 Conversion based on one mode of vibration

To be completed by Anirudh.

4.4.2 Conversion using an adaptive approach

To be completed by Anirudh.

4.5 Nonlinear static procedures with record-to-record dispersion

Ruiz-Garcia and Miranda (2007), Vamvatsikos and Cornell (2006) and Dolsek and Fajfar (2004) studies on the assessment of nonlinear structural response, have been integrated in three nonlinear static procedures, which are based on the use of capacity curves, resulting from nonlinear static pushover analysis, to determine directly the median seismic intensity values $\hat{S}_a$ corresponding to the attainment of a certain damage state threshold (limit state) and the corresponding dispersion β_{S_a}. These parameters are used to represent a fragility curve as the probability of the limit state capacity C being exceeded by the demand D, both expressed in terms of intensity levels ($S_{a,ds}$ and S_a respectively), as shown in the following equation:

$$P_{LS}(S_a) = P(C < D|S_a) = \Phi(\frac{lnS_a - ln\hat{S}_{a,ds}}{\beta_{S_a}}) \tag{4.1}$$

The methodologies implemented allow to consider different shapes of the pushover curve, multilinear and bilinear, record-to-record dispersion and dispersion in the damage state thresholds in a systematic and harmonised way.

The intensity measure to be used is S_a and a mapping between any engineering demand parameter (EDP), assumed to describe the damage state thresholds, and the roof displacement should be available from the pushover analysis.

The methodologies are originally built for single building fragility curves, however the fragility curves derived for single buildings can be combined in a unique fragility curve, which considers also the inter-building uncertainty, as described in the Section 4.5.1.

4.5.1 SPO2IDA (Vamvatsikos and Cornell 2006)

The tool spo2ida (**VamvatsikosCornell2005**) is capable of converting static pushover curves into 16%, 50% and 84% ida curves, as shown in Figure 4.1, using empirical relationships from a large database of incremental dynamic analysis results.

The spo2ida tool is applicable to any kind of multi-linear capacity curve. Making use of this tool is is possible to estimate single building fragility curve and fragility curves derived for single buildings can be combined in a unique fragility curve, which considers also the inter-building uncertainty.

Given an idealised capacity curve the spo2ida tool uses an implicit R-μ-T relation to correlate nonlinear displacement, expressed in terms of ductility μ to the corresponding median capacities in terms of the parameters R. R is the lateral strength ratio, defined as the ratio between the spectral acceleration S_a and the yielding capacity of the system S_{ay}. Each branch of the capacity curve, hardening, softening and residual plateau, is converted to a corresponding branch of the three ida curves, using the R-μ-T relation, which is a function of the hardening stiffness, the softening stiffness and the residual force. These parameters are derived from the idealised pushover

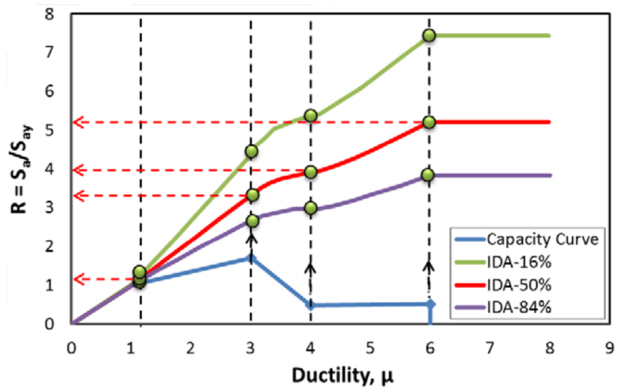

Figure 4.1 – *spo2ida tool: IDA curves derived from Pushover curve.*

capacity expressed in μ-R terms, as well as the ductility levels at the onset of each branch. If some of the branches of the pushover curve are missing because of the seismic behaviour of the system, spo2ida can equally work with bilinear, trilinear and quadrilinear idealisations.

The result of the spo2ida routine is thus a list of ductility levels and corresponding R values at 50%, 16% and 84% percentiles. For any inelastic displacement, and therefore any level of ductility μ the corresponding $R_{50\%}$, $R_{16\%}$, and $R_{84\%}$ values are found interpolating the aforementioned ida curves. Median R and its dispersion at ductility levels corresponding to the damage thresholds ds can thus be determined, and converted into median $S_{a,ds}$ and its dispersion due to record-to-record variability $\beta_{S_{ad}}$ according to equations 4.2 and 4.3.

$$\hat{S}_{a,ds} = R_{50\%}(\mu_{ds})S_{ay} \tag{4.2}$$

$$\beta_{S_{ad}} = \beta_{R(\mu_{ds})} = \frac{\ln R(\mu_{ds})_{84\%} - \ln R(\mu_{ds})_{16\%}}{2} \tag{4.3}$$

If dispersion due to uncertainty in the limit state definition $\beta_{\theta c}$ is different from zero a Monte Carlo sampling needs to be performed to combine it with the record-to-record dispersion. Different values of ductility limit state are sampled from the lognormal distribution with median the median value of the ductility limit state, and dispersion the input $\beta_{\theta c}$. For each of these ductilities the corresponding median $R_{50\%}$ and $R_{16\%}$, $R_{84\%}$ are found and converted into $\hat{S}_{a,ds}$ and β_{Sad} according to equation 4.2 and 4.3. MC random S_a for each MC sampled ductility limit state are computed, and their median and the dispersion are estimated. These parameters constitute the median $\hat{S}_{a,ds}$ and the total dispersion β_{S_a} for the considered damage state. The procedure is repeated for each damage state.

4.5.1.1 Multiple-Building Fragility and Vulnerability function

If multiple buildings have been input to derive fragility function for a class of buildings all $\hat{S}_{a,blg}$ and $\beta_{S_a,blg}$ are combined in a single lognormal curve. A minimum of 5 buildings should be considered to obtain reliable results for the class.

A new issue arises when multiple buildings are considered: the S_a at the fundamental period of each building should be converted to a common intensity measure, to be able to combine the different fragility functions. A common intensity measure is selected to be S_a at the period T_{av}, which is a weighted average of the individual building fundamental periods T_1. Then each individual fragility needs to be expressed in terms of the common $S_a(T_{av})$, using a spectrum. FEMA P-695 (**ATC2007**) far field set of 44 accelerograms (22 records for the two directions) was used to derive a mean uniform hazard spectrum, and the ratio between the S_a at different periods is used to scale the fragility functions. It can be noted that the actual values of the spectrum are not important, but just the spectral shape. The median $\hat{S}_a$ is converted to the mean $\mu_{ln(S_a)}$ of the corresponding normal distribution ($\mu_{ln(S_a)} = ln(\hat{S}_a)$) and simply scaled to the common intensity measure as follows:

$$\mu_{ln(S_a),blg} = \mu_{ln(S_a),blg} S(T_{av})/S(T_{1,blg}) \tag{4.4}$$

$$\beta_{S_a,blg} = \beta_{S_a,blg} S(T_{av})/S(T_{1,blg}) \tag{4.5}$$

where $S(T_{av})/S(T_{1,blg}$ is defined as spectral ratio. Finally the parameters of the single lognormal curve for the class of buildings, mean and dispersion, can be computed as the weighted mean of the single means and the weighted SRSS of the inter-building and intra-building standard deviation, the standard deviation of the single means and the single dispersions respectively, as shown in the following equations:

$$\mu_{ln(S_a),tot} = \sum_{i=0}^{n.blg} w_{blg-i} \mu_{ln(S_a),blg-i} \tag{4.6}$$

$$\beta_{S_a,tot} = \sqrt{\sum_{i=0}^{n.blg} w_{blg-i}((\mu_{ln(S_a),blg-i} - \mu_{ln(S_a),tot})^2 + \beta^2_{S_a,blg-i})} \tag{4.7}$$

In order to use this methodology, it is necessary to load one or multiple capacity curves as described in Section 4.2.1. It is also necessary to specify the type of shape the capacity curves want to be idealised with, using the parameter `idealised_type` (either `bilinear` or `quadrilinear`). If the user has already at disposal an idealised multilinear pushover curve for each building, the variable `Idealised` in the csv input file should be set to `TRUE`, and idealised curves should be provided according to what described in section 4.2.1. Then, it is necessary to specify a damage model using the parameter `damage_model` (see Section 4.2.3).

If dispersion due to uncertainty in the limit state definition is different from zero a Monte Carlo sampling needs to be performed to combine it with the record-to-record dispersion. The number of Monte Carlo samples should be defined in the variable `montecarlo_samples`. After importing the module `SPO2IDA_procedure`, it is possible to calculate the parameter of the fragility model, median and dispersion, using the following command:

```
fragility_model = SPO2IDA_procedure.calculate_fragility(capacity_curves,
... idealised_capacity, damage_model, montecarlo_samples, Sa_ratios,
... ida_plotflag)
```

where `Sa_ratios` is the spectral ratio variable needed to combine together fragility curves for many buildings, as described in Section 4.7.1, and `ida_plotflag` indicates whether ida plots want to be displayed (`ida_plotflag` = 1) or not (`ida_plotflag` = 0).

4.5.2 Dolsek and Fajfar 2004

This procedure by (**DolsekFajfar2004**) provides a simple relationship between inelastic displacement of a SDoF system and the corresponding median elastic spectral displacement value. The procedure presented herein is applicable to any kind of multi-linear capacity curve and it can be used to estimate single building fragility curve. Moreover the fragility curves derived for single buildings can be combined in a unique fragility curve, which considers also the inter-building uncertainty.

The relationship provided by (**DolsekFajfar2004**) has been adapted for MDoF systems, relating the inelastic top displacement of a structure $\hat{d}_{roof}$ to the median elastic spectral acceleration value at its fundamental period of vibration $\hat{S}_a(T_1)$, as presented in the following equation:

$$\hat{S}_a(T_1) = \frac{4\pi^2}{\hat{C}_R T^2 \Gamma_1 \Phi_1} \hat{d}_{roof} \tag{4.8}$$

where $\Gamma_1\Phi_1$ is the first mode participation factor estimated for the first-mode shape normalised by the roof displacement. The value of C_R, the ratio between the inelastic and the elastic spectral displacement, is found from the following equation:

$$\hat{C}_R = \frac{\mu}{R(\mu)} \tag{4.9}$$

where μ and R are the median values of ductility level and the reduction factor for that level of ductility respectively. R is defined as the ratio between the spectral acceleration S_a and the yielding capacity of the system S_{ay}. According to the results of an extensive parametric study using three different sets of recorded and semi-artificial ground motions, (**DolsekFajfar2004**) related the ductility demand μ and reduction factor R through the following formula:

$$\mu = \frac{1}{c}(R - R_0) + \mu_0 \tag{4.10}$$

In the proposed model μ is linearly dependent on R within two reduction factor intervals. The parameter c defines the slope of the R–μ relation, and it depends on the idealised pushover curve parameters (the initial period of the system T, the maximum to residual strength ratio r_u, the ductility at the onset of degradation μ_s) and the corner periods T_c and T_d. T_c and T_d are the corner periods between the constant acceleration and constant velocity part of the idealised elastic spectrum, and between the constant velocity and constant displacement part of the idealised elastic spectrum respectively. R_0 and μ_0 are the values of R and μ on the capacity curve corresponding to the onset of hardening or softening behaviour, according to the following relationship:

$$\mu_0 = 1 ... if R <= R(\mu_s); \mu_0 = \mu_s ... if R > R(\mu_s) \tag{4.11}$$

$$R_0 = 1 ... if R <= R(\mu_s); R_0 = R(\mu_s) ... if R > R(\mu_s) \tag{4.12}$$

Given the parameters of the multilinear pushover curves (R_0, μ_0, r_u, μ_s) and T, the median R-μ curve can be constructed using the aforementioned relationship, as presented in the following Figure.

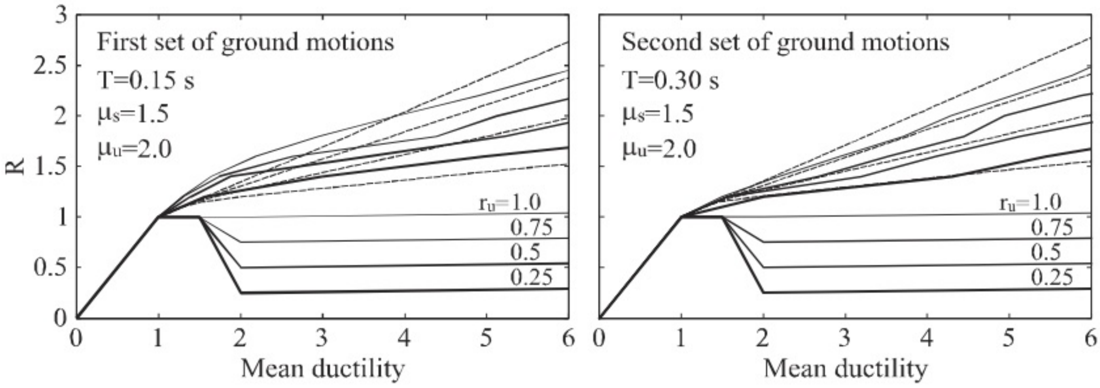

Figure 4.2 – *R-μ curves derived from Pushover curve.*

The relationship between the 16^{th} and 84^{th} fractiles of μ and R_{50} needs to be derived using the equations from (**RuizGarciaMiranda2007**) instead, given that (**DolsekFajfar2004**) do not provide estimates of the dispersion of R. This is done by computing the value of record-to-record dispersion in terms of top displacement $\beta_{\theta d}$ for a number of R with eq. 4.19, and calculating the 16^{th} and 84^{th} fractiles of μ ($\mu_{16\%}$ and $\mu_{84\%}$), according to the Equations 4.13 and 4.14. The $\mu_{50\%} - R_{50\%}$, $\mu_{16\%} - R_{50\%}$ and $\mu_{84\%} - R_{50\%}$ curves can thus be drawn, as shown in Figure 4.3.

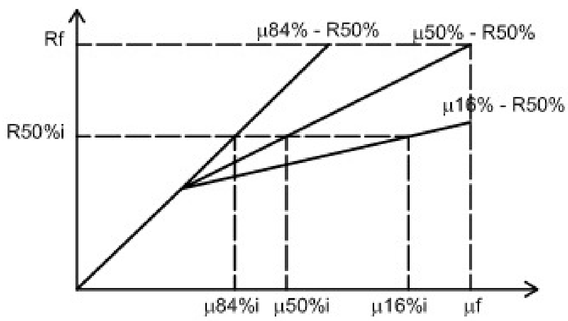

Figure 4.3 – $\mu_{50\%} - R_{50\%}$, $\mu_{16\%} - R_{50\%}$ *and* $\mu_{84\%} - R_{50\%}$ *curves*

$$\mu_{ds,16} = \hat{\mu}_{ds} e^{-\beta_{\theta d,ds}} \tag{4.13}$$

$$\mu_{ds,84} = \hat{\mu}_{ds} e^{\beta_{\theta d,ds}} \tag{4.14}$$

For any inelastic displacement, and therefore any level of ductility μ the corresponding $R_{50\%}$, $R_{16\%}$, and $R_{84\%}$ values are found interpolating the aforementioned curves. Median R and its dispersion at ductility levels corresponding to the damage thresholds ds can thus be determined, and converted into median $S_{a,ds}$ and its dispersion due to record-to-record variability $\beta_{S_{ad}}$ according to equations 4.2 and 4.3.

If dispersion in the damage state threshold is different from zero, different values of ductility limit state are sampled from the lognormal distribution with median the median value of the ductility limit state, and dispersion the input $\beta_{\theta c}$. For each of these ductilities the corresponding $R_{50\%}$, $R_{16\%}$, and $R_{84\%}$ values are found interpolating the $\mu_{50\%} - R_{50\%}$, $\mu_{16\%} - R_5 0\%$ and $\mu_{84\%} - R_5 0\%$ curves, and converted into $\hat{S}_{a,ds}$ and $\beta_{S_{ad}}$ according to Equations 4.2 and 4.3. MC random S_a for each of the MC sampled ductility limit states are computed using $\hat{S}_{a,ds}$ and $\beta_{S_{ad}}$, and their median and dispersion are estimated. These parameters constitute the median $\hat{S}_{a,ds}$ and the total dispersion β_{S_a} for the considered damage state. The procedure is repeated for each damage state.

If multiple buildings have been input to derive fragility function for a class of buildings all $\hat{S}_{a,blg}$ and $\beta_{S_a,blg}$ are combined in a single lognormal curve as described in section 4.5.1.

In order to use this methodology, it is necessary to load one or multiple capacity curves as described in Section 4.2.1. The capacity curves are then idealised with a bilinear elasto-plastic shape. It is also necessary to specify the type of shape the capacity curves want to be idealised with, using the parameter `idealised_type` (either `bilinear` or `quadrilinear`). If the user has already at disposal an idealised multilinear pushover curve for each building, the variable `Idealised` in the csv input file should be set to `TRUE`, and idealised curves should be provided according to what described in section 4.2.1. Then, it is necessary to specify a damage model using the parameter `damage_model` (see Section 4.2.3).

If dispersion due to uncertainty in the limit state definition is different from zero a Monte Carlo sampling needs to be performed to combine it with the record-to-record dispersion. The number of Monte Carlo samples should be defined in the variable `montecarlo_samples`. After importing the module DF2004, it is possible to calculate the parameter of the fragility model, median and dispersion, using the following command:

```
fragility_model = DF2004.calculate_fragility(capacity_curves, ...
idealised_capacity, damage_model, montecarlo_samples, Sa_ratios, ...
corner_periods)
```

where `Sa_ratios` is a variable needed to combine together fragility curves for many buildings, as described in Section 4.5.1.

4.5.3 Ruiz Garcia and Miranda 2007

The research by (**RuizGarciaMiranda2007**) provides a simple relationship for SDoF systems between inelastic displacement and the corresponding median elastic spectral displacement value. The procedure presented herein is applicable to bilinear elasto-plastic capacity curve only and it can be used to estimate single building fragility curve. Moreover the fragility curves derived for single buildings can be combined in a unique fragility curve, which considers also the inter-building uncertainty.

The relationship provided by (**RuizGarciaMiranda2007**) has been adapted for MDoF systems ((**Vamvatsikos2014**)), relating the inelastic top displacement of a structure $\hat{d}_{roof}$ to the median elastic spectral displacement value at its fundamental period of vibration $\hat{S}_d(T)$, as presented in the following equation:

$$\hat{S}_d(T_1) = \frac{\hat{d}_{roof}}{\hat{C}_R \Gamma_1 \Phi_1} \tag{4.15}$$

where $\Gamma_1\Phi_1$ is the first mode participation factor estimated for the first-mode shape normalised by the roof displacement, and C_R is the inelastic displacement ratio (inelastic over elastic spectral displacement), computed by (**RuizGarciaMiranda2007**) for nonlinear SDoF systems, which is a function of the first-mode period of vibration and the relative lateral strength of the system R. Therefore the median spectral acceleration at the fundamental period of vibration $\hat{S}_a(T)$ turns out to be expressed as a function of top displacement according to the following equation:

$$\hat{S}_a(T) = \frac{4\pi^2}{\hat{C}_R T^2 \Gamma_1 \Phi_1} \hat{d_{roof}} \tag{4.16}$$

Estimates of $\hat{C}_R$ parameter are provided by (**RuizGarciaMiranda2007**), as result of non-linear regression analysis of three different measures of central tendency computed from 240 ground motions:

$$\hat{C}_R = 1 + \frac{\hat{R}-1}{79.12 T_1^{1.98}} \tag{4.17}$$

where $\hat{R}$ is given by the following equation:

$$\hat{R} = max(0.425(1-c+\sqrt{c^2+2c(2\hat{\mu}-1)+1}),1) \tag{4.18}$$

where c = 79.12 $T^{1.98}$, and $\hat{\mu}$ is the median ductility level of interest.

Moreover (**RuizGarciaMiranda2007**) provide an estimate of the dispersion of C_R parameter due to record-to-record variability with Equation 4.19, that can be assumed equal to the dispersion of d_{roof}, since the two quantities are proportional.

$$\sigma_{\ln(C_R)} = \sigma_{\ln(d_{roof})} = \beta_{\theta d} = 1.975[\frac{1}{5.876} + \frac{1}{11.749(T+0.1)}][1-\exp(-0.739(R-1))] \tag{4.19}$$

The median value of S_a corresponding to any level of ductility (d_{roof}/d_y) can be defined combining Equation from 4.16 to 4.18. The relationship between the 16^{th} and 84^{th} fractiles of μ and R can be drawn instead, computing $\beta_{\theta d}$ for a discretised number of R with eq. 4.19, and calculating the 16^{th} and 84^{th} fractiles of μ ($\mu_{16\%}$ and $\mu_{84\%}$), according to the Equations 4.13 and 4.14. $\mu_{16\%}$-$R_{50\%}$, $\mu_{50\%}$-$R_{50\%}$ obtained in such way are shown in Figure 4.4.

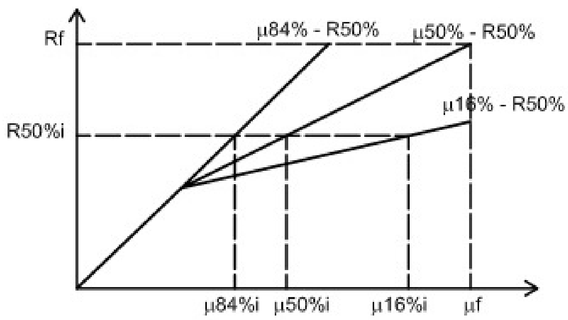

Figure 4.4 – *R-μ relationship.*

For any inelastic displacement, and therefore any level of ductility μ the corresponding $R_{50\%}$, $R_{16\%}$, and $R_{84\%}$ values are found interpolating the aforementioned curves. Median R and its dispersion at ductility levels corresponding to the damage thresholds ds can thus be determined, and converted into median $S_{a,ds}$ and its dispersion due to record-to-record variability $\beta_{S_{ad}}$, according to equations 4.2 and 4.3.

If dispersion in the damage state threshold is different from zero, different values of ductility limit state are sampled from the lognormal distribution with median the median value of the ductility limit state, and dispersion the input $\beta_{\theta c}$. For each of these ductilities the corresponding $R_{50\%}$, $R_{16\%}$, and $R_{84\%}$ values are found interpolating the $\mu_{50\%} - R_{50\%}$, $\mu_{16\%} - R_50\%$ and $\mu_{84\%} - R_50\%$ curves, and converted into $\hat{S}_{a,ds}$ and $\beta_{S_{ad}}$ according to Equations 4.2 and 4.3. MC random S_a for each of the MC sampled ductility limit states are computed using $\hat{S}_{a,ds}$ and $\beta_{S_{ad}}$, and their median and dispersion are estimated. These parameters constitute the median $\hat{S}_{a,ds}$ and the total dispersion β_{S_a} for the considered damage state. The procedure is repeated for each damage state.

If multiple buildings have been input to derive fragility function for a class of buildings all $\hat{S}_{a,blg}$ and $\beta_{S_a,blg}$ are combined in a single lognormal curve as described in section 4.5.1.

In order to use this methodology, it is necessary to load one or multiple capacity curves as described in Section 4.2.1. The capacity curves are then idealised with a bilinear elasto-plastic shape. If the user has already at disposal an idealised multilinear pushover curve for each building, the variable `Idealised` in the csv input file should be set to `TRUE`, and idealised curves should be provided according to what described in section 4.2.1. Then, it is necessary to specify a damage model using the parameter `damage_model` (see Section 4.2.3).

If dispersion due to uncertainty in the limit state definition is different from zero a Monte Carlo sampling needs to be performed to combine it with the record-to-record dispersion. The number of Monte Carlo samples should be defined in the variable `montecarlo_samples`. After importing the module RGM2007, it is possible to calculate the parameter of the fragility model, median and dispersion, using the following command:

```
fragility_model = RGM2007.calculate_fragility(capacity_curves, ...
idealised_capacity, damage_model, montecarlo_samples, Sa_ratios)
```

where `Sa_ratios` is the spectral ratio variable, needed to combine together fragility curves for many buildings, as described in Section 4.5.1.

4.6 Assessment of nonlinear structural response

4.6.1 Vidic and Fajfar 1994

This procedure aims to determine the displacements from an inelastic spectra for systems with a given ductility factor. The inelastic displacement spectra is determined by means of applying a ductility-based reduction factor (C), which depends on the natural period of the system, the given ductility factor, the hysteretic behaviour, the damping model, and the frequency content of the ground motion.

The procedure proposed by (**VidicEtAl1994**) was validated by a comparison of the approximate spectra with the "exact" spectra obtained from non-linear dynamic time history analyses. Records from California and Montenegro were used as representative of "standard" ground motion, while the influence of input motion was analysed using other five groups of records (coming from different parts of the world) that represented different types of ground motions. The influence of the hysteretic models was taken into account by considering the bilinear model and the stiffness degrading Q-model. Finally, in order to analyse the effect of damping, two models were considered: "mass-proportional" damping, which assumes a time-independent

Table 4.1 – *Paramereters for the estimation of the reduction factor C proposed by (***VidicEtAl1994***)*

Hysteresis model	Damping model	c_1	c_2	c_R	c_T
Q	Mass	1.00	0.65	1.00	0.30
Q	Stiffness	0.75	0.65	1.00	0.30
Bilinear	Mass	1.35	0.75	0.95	0.20
Bilinear	Stiffness	1.10	0.75	0.95	0.20

damping coefficient based on elastic properties, and "instantaneous stiffness-proportional" damping, which assumes a time-dependent damping coefficient based on tangent stiffness. For most cases, a damping ratio of 5% was assumed, although for some systems a value of 2% was adopted.

It is possible to derive approximate strength and displacement inelastic spectra from an elastic pseudo-acceleration spectrum using the proposed modified spectra. In the medium and long-period region, it was observed that the reduction factor is slightly dependant on the period T and is roughly equal to the prescribed ductility (μ). However, in the short-period region, the factor C strongly depends on both T and μ. The influence of hysteretic behaviour and damping can be observed for the whole range of periods. Based on this, a bilinear curve was proposed. Starting in C = 1, the value of C increases linearly along the short-period region up to a value approximately equal to the ductility factor. In the medium- and long-period range, the C-factor remains constant. This is mathematically expressed by the following relationships:

$$C_\mu = \begin{cases} c_1\,(\mu-1)^{c_R}\,\frac{T}{T_0}+1, & T \le T_0 \\ c_1\,(\mu-1)^{c_R}+1 & T > T_0 \end{cases} \tag{4.20}$$

where:

$$T_0 = c_2 \mu^{c_T} T_c \tag{4.21}$$

And T_c stands for the characteristic spectral period and c_1, c_2, c_R, c_T are constants dependant on the hysteretic behaviour and damping model, as defined in Table 4.1.

In order to use this methodology, it is necessary to load one or multiple capacity curves as described in Section 4.2.1, as well as a set of ground motion records as explained in Section 4.2.2. Then, it is necessary to specify a damage model using the parameter `damage_model` (see Section 4.2.3), and a damping ratio using the parameter `damping`. It is also necessary to specify the type of hysteresis (`Q` or `bilinear`) and damping (`mass` or `stiffness`) models as defined in Table 4.1, using the parameters `hysteresis_model` and `damping_model`, respectively. After importing the module `vidic_etal_1994`, it is possible to calculate the distribution of structures across the set of damage state for each ground motion record using the following command:

```
PDM, Sds = vidic_etal_1994.calculate_fragility(capacity_curves,gmrs,...
damage_model,damping)
```

Where `PDM` (i.e. probability damage matrix) represents a matrix with the number of structures in each damage state per ground motion record, and `Sds` (i.e. spectral displacements) represents a matrix with the maximum displacement (of the equivalent SDOF) of each structure per ground motion record. the variable PDM can then be used to calculate the mean fragility model as described in Section 4.7.1.

Table 4.2 – *Paramereters for the estimation of the reduction factor C proposed by (***LinMiranda2008***)*

α	m_1	m_2	n_1	n_2
0%	0.026	0.87	0.016	0.84
5%	0.026	0.65	0.027	0.55
10%	0.027	0.51	0.031	0.39
20%	0.027	0.36	0.030	0.24

4.6.2 Lin and Miranda 2008

This methodology estimates the maximum inelastic displacement of an existing structure based on the maximum elastic displacement response of its equivalent linear system without the need for iterations, based on the strength ratio R (instead of the most commonly used ductility ratio).

In order to evaluate an existing structure, a pushover analysis should be conducted in order to obtain the capacity curve. This curve should be bilinearised in order to obtain the yield strength, fy, the postyield stiffness ratio, α, and the strength ratio, R. With these parameters, along with the initial period of the system, it is possible to estimate the optimal period shift (i.e. the ratio between the period of the equivalent linear system and the initial period) and the equivalent viscous damping, ξeq, of the equivalent linear system, using the following relationships derived by (**LinMiranda2008**).

$$\frac{T_{eq}}{T_0} = 1 + \frac{m_1}{T_0^{m_2}} \left(R^{1.8} - 1\right) \tag{4.22}$$

$$\xi_{eq} = \xi_0 + \frac{n_1}{T_0^{n_2}} (R - 1) \tag{4.23}$$

Where the coefficients m1, m2, n1, and n2 depend on the postyield stiffness ratio, as shown in the following Table 4.2.

Using ξeq and the damping modification factor, B (as defined in Table 15.6-1 of NEHRP-2003), it is possible to construct the reduced displacement spectrum, Sd(T, ξeq) from which the maximum displacement demand (i.e. the displacement corresponding to the equivalent system period) can be obtained, using the following equation:

In order to use this methodology, it is necessary to load one or multiple capacity curves as described in Section 4.2.1, as well as a set of ground motion records as explained in Section 4.2.2. Then, it is necessary to specify a damage model using the parameter `damage_model` (see Section 4.2.3). After importing the module `lin_miranda_2008`, it is possible to calculate the distribution of structures across the set of damage state for each ground motion record using the following command:

```
PDM, Sds = lin_miranda_2008.calculate_fragility(capacity_curves,gmrs,...
damage_model,damping)
```

Where `PDM` (i.e. probability damage matrix) represents a matrix with the number of structures in each damage state per ground motion record, and `Sds` (i.e. spectral displacements) represents a matrix with the maximum displacement (of the equivalent SDOF) of each structure per ground motion record. the variable PDM can then be used to calculate the mean fragility model as described in Section 4.7.1.

4.6.3 Miranda (2000) for firm soils

This study by **Miranda2000** aims to quantify the influence of soil conditions, earthquake magnitude, and epicentral distance on the inelastic displacement ratios, C_μ. For two systems with the same mass and period of vibration that have been subjected to the same earthquake ground motion. C_μ can be defined as the ratio of the maximum lateral inelastic displacement demand of one to the maximum lateral elastic displacement demand on the other, as shown in the following equation:

$$C_\mu = \frac{\Delta_{inelastic}}{\Delta_{elastic}} \tag{4.24}$$

In this study, 264 earthquake acceleration time histories recorded in California (USA) for 12 different events were used. In order to investigate the effect of the soil conditions, the records were classified into three groups: the first one consisted of ground motions recorded on stations located on rock (i.e. average shear-wave velocities >760 m/s). The second group included the records registered on stations on very dense soil or soft rock (i.e. average shear-wave velocities between 360 m/s and 760 m/s). Finally, the third group consisted of ground motion records from stations located on stiff soil (i.e. average shear-wave velocities between 180 m/s and 360 m/s).

It was observed that for periods longer than about 1.0 s, the mean inelastic displacement ratios are approximately equal to 1, meaning that, on average, the maximum inelastic displacements are equal to the maximum inelastic displacements. On the other hand, for periods smaller than 1.0 s, the mean inelastic displacement ratios are larger than 1 and strongly depend on the period of vibration and on the level of inelastic deformation. The results of the investigation yielded that for the sites under consideration (i.e. average shear-wave velocities higher than 180 m/s) neither the soil conditions, nor the earthquake magnitude, nor the distance to rupture cause significant differences on the value of C_μ. However, if directivity effects are taken into consideration, the inelastic displacement ratios for periods between 0.1 s and 1.3 s can be larger than those estimated for systems not affected by directivity.

Based on the results of the mean inelastic displacement ratios, nonlinear regression analyses were conducted to estimate the following simplified expression, which allows estimating the inelastic displacement ratio of a system:

$$C_\mu = \left[1 + \left(\frac{1}{\mu} - 1\right) exp\left(-12T\mu^{-0.8}\right)\right]^{-1} \tag{4.25}$$

Where μ = displacement ductility ratio and T = period of vibration.

In order to use this methodology, it is necessary to load one or multiple capacity curves and a set of ground motio nrecords, as explained in Section 4.2.1 and 4.2.2, respectively. Then, it is necessary to specify a damage model using the parameter `damage_model` (see Section 4.2.3), and a damping ratio using the parameter `damping`. After importing the module `miranda_2000_firm_soils`, it is possible to calculate the distribution of structures across the set of damage states for each ground motion record using the following command:

```
PDM, Sds = miranda_2000_firm_soils.calculate_fragility(capacity_curves,...
gmrs,damage_model,damping)
```

Where `PDM` (i.e. probability damage matrix) represents a matrix with the number of structures in each damage state per ground motion record, and `Sds` (i.e. spectral displacements) represents a matrix with the maximum displacement (of the equivalent SDOF) of each structure per ground motion record. the variable PDM can then be used to calculate the mean fragility model as described in Section 4.7.1.

4.6.4 N2 (EC8, CEN 2005)

This simplified nonlinear procedure has been firstly proposed by Fajfar, and it is capable of estimating the seismic response of structures using capacity curves (for the equivalent SDOF) and response spectra. It is somehow similar to the well-known Capacity Spectrum Method (see Section), but it does not require an iterative process and instead of elastic overdamped spectra, it uses inelastic response spectra. This method is part of recommendations of the Eurocode 8 (**CEN2005**) for the seismic design of new structues, and the capacity curves are usually simplified by a elasto-perfectly plastic relationship.

To estimate the target displacement (δ_t) within this methodology, it is necessary to assess whether the SDOF structure is in the short-period or medium/long-period ranges. To do so, it is necessary to compare the fundamental period of vibration of the structure with the corner period of the ground motion record. If the structure is in the latter category, it is assumed that the target displacement is equal to the elastic spectral displacement for the fundamental period of the idealized SDOF. If on the other hand it is located in the short-period range, a procedure is carried out to evaluate whether the capacity of the SDOF at the yielding point (taken from the bilinear curve) is lower than the spectral acceleration response for the same period. If this is verified, then the structure is assumed to have an elastic response and once again, the target displacement will be equal to the elastic spectral displacement for the fundamental period. In case the capacity is lower than the response for the yielding point, the structure is assumed to have an inelastic response and the following formula is employed to determine the target displacement:

$$\delta_t = \frac{Sd(T_{el})}{q_u}\left(1+(q_u-1)\frac{T_c}{T_{el}}\right)Sd(T_{el}) \tag{4.26}$$

Where $Sd(T_{el})$ stands for the spectral displacement for the fundamental period of the idealized SDOF (T_{el}), T_c stands for the corner period and q_u represents the ratio between the spectral acceleration for T_{el} and the acceleration at the yielding point.

It is important to understand that this methodology has been developed originally to be combined with a design or code-based response spectrum, and not with a spectrum derived from real ground motion records. For this reason, its employment in the derivation of fragility functions calls for due care. For instance, the estimation of T_c (which is a fundamental parameter within this methodology) when considering accelerograms is not a trivial task, and various proposals can be found in the literature. For the sake of simplicity, a decision was made to adopt the formula recommended by the **ASCE2010** which defines:

$$T_c = \frac{Sa(T=1.0s)}{Sa(T=0.2s)} \tag{4.27}$$

Despite these caveats, it is worth mentioning that a recent study (**SilvaEtAl2014b**) compared its performance in the derivation of vulnerability functions against nonlinear time history analyses, and concluded that reasonable results can still be obtained.

In order to use this methodology, it is necessary to load one or multiple capacity curves and a set of ground motio nrecords, as explained in Section 4.2.1 and 4.2.2, respectively. Then, it is necessary to specify a damage model using the parameter `damage_model` (see Section 4.2.3), and a damping ratio using the parameter `damping`. After importing the module `N2Method`, it is possible to calculate the distribution of structures across the set of damage states for each ground motion record using the following command:

```
PDM, Sds = N2Method.calculate_fragility(capacity_curves,gmrs,...
damage_model,damping)
```

Where `PDM` (i.e. probability damage matrix) represents a matrix with the number of structures in each damage state per ground motion record, and `Sds` (i.e. spectral displacements) represents a matrix with the maximum displacement (of the equivalent SDOF) of each structure per ground motion record. the variable PDM can then be used to calculate the mean fragility model as described in Section 4.7.1.

4.6.5 Capacity Spectrum Method (FEMA, 2005)

The capacity spectrum method (CSM) was initially proposed by (**FreemanEtAl1975**), and it represents a simplified methodology for many purposes such as the evaluation of a large inventory of buildings, structural assessment of new or existing buildings or to identify the correlation between damage states and levels of ground motion. **ATC1996** proposes three different procedures (A, B and C) for the application of the Capacity Spectrum Method. However, procedure B adopts some simplifications that might not always be valid and procedure C has a very strong graphical component, making it difficult for systematic applications. Hence, procedure A, which is characterized by its intuitiveness and simplicity, has been implemented in the RMTK.

This procedure iteratively compares the capacity and the demand of a structure, using a capacity curve (for the equivalent SDOF) and a damped response spectrum, respectively. The ground motion spectrum is computed for a level of equivalent viscous damping that is estimated as a function of the displacement at which the response spectrum crosses the capacity curve, in order to take into account the inelastic behaviour of the structure. Iterations are needed until there is a match between the equivalent viscous damping of the structure and the damping applied to the spectrum. The final intersection of these two curves approximates the target displacement response of the structure. This result is presented in Figure 4.5 for a "weak" and a "strong" ground motion record.

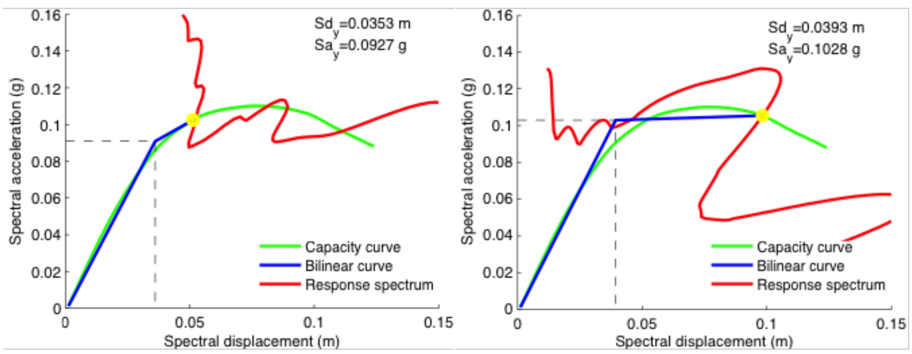

Figure 4.5 – *Assessment of the target displacement for "weak" (left) and "strong" (strong) ground motion record.*

The initial proposal of this method was heavily criticized due to its tendency to underestimate the deformation of the structures, which was mostly related with the model employed to calculate the equivalent viscous damping (e.g. **Fajfar1999 ChopraGoel2010**). Thus, in **FEMA4402005** some modifications were proposed regarding the calculation of this component. Furthermore, several other models relating an equivalent viscous damping ratio (ξ_{eq}) with a ductility level (μ) have been proposed in the last decades, and implemented in the RMTK. The following list describes these models, and specifies the code that must be defined in the variable `damping_model` in order to follow the associated model in the vulnerability calculations.

- **FEMA4402005** This model assumes different expressions to calculate the equivalent viscous damping ratio depending on the ductulity level, hysteretic model and post-elastic stiffness. However, for the sake of simplicity, aproximate equations have been proposed to calculate ξ_{eq} with any capacity curve, that only depends on the level of ductility, as described below:

  For 1.0< μ <4.0:
  $$\xi_{eq} = \xi_0 + 0.049\,(\mu - 1)^2 - 0.01\,(\mu - 1)^3 \tag{4.28}$$
  For 4.0≤ μ ≤6.5:
  $$\xi_{eq} = \xi_0 + 0.14 + 0.0032\,(\mu - 1) \tag{4.29}$$
  For μ >6.5:
  $$\xi_{eq} = \xi_0 + 0.19\left[\frac{0.64\,(\mu - 1) - 1}{[0.64\,(\mu - 1)]^2}\right]\left(\frac{T_{eff}}{T_0}\right)^2 \tag{4.30}$$
  Where ξ_0 stands the initial elastic viscou damping ratio, and T_{eff} represents for the effective period, which for ductility above 6.5 can be calculated using the following expression:
  $$T_{eff} = \left\{0.89\left[\sqrt{\frac{(\mu - 1)}{1 + 0.05(\mu - 2)}}\right] + 1\right\} T_0 \tag{4.31}$$
  In order to use this model, the variable `damping_model` must be set to `FEMA_2005`.

- **Kowalsky1994** This model estblishes a relationship between the equivlent viscous damping ratio and a ductility level and a post-yield stifness ratio α, as defined by the following equation:
  $$\xi_{eq} = \xi_0 + \frac{1}{\pi}\left[1 - \frac{(1 - \alpha)}{\sqrt{\mu}} - \alpha\sqrt{\mu}\right] \tag{4.32}$$
  In order to use this model, the variable `damping_model` must be set to `Kowalsky_1994`.

- (**Iwan1980**): This model was developed using a limited number of ground motion records and a single hysteretic model, leading to the following equation:
  $$\xi_{eq} = \xi_0 + 0.0587\,(\mu - 1)^0.371 \tag{4.33}$$
  In order to use this model, the variable `damping_model` must be set to `Iwan_1980`.

- **GulkanSozen1974**This model was derived considering the Takeda hysteretic for elasto-plastic systems calibrated with experimental shaking-table results of a number of reinforced concrete frames. The equivalent viscous damping ratio is calculated using the following ductility-dependent formula:

$$\xi_{eq} = \xi_0 + 0.2\left(1 - \frac{1}{\sqrt{\mu}}\right) \tag{4.34}$$

In order to use this model, the variable `damping_model` must be set to `Gulkan_Sozen_1974`.

- **PriestleyEtAl2007** These Authors proposed different models depending on the structure type. Currently, three models proposed by this study have been implemented, as described below:

For reinforced concrete frame structures:

$$\xi_{eq} = 0.05 + 0.565\left(\frac{\mu - 1}{\pi\mu}\right) \tag{4.35}$$

To use this model set the variable `damping_model` to `Priesley_et_al2007_frames`.

For reinforced concrete walls structures:

$$\xi_{eq} = 0.05 + 0.444\left(\frac{\mu - 1}{\pi\mu}\right) \tag{4.36}$$

To use this model set the variable `damping_model` to `Priesley_et_al2007_walls`.

For steel structures:

$$\xi_{eq} = 0.05 + 0.577\left(\frac{\mu - 1}{\pi\mu}\right) \tag{4.37}$$

To use this model set the variable `damping_model` to `Priesley_et_al2007_steel`.

- **Calvi1999** This Author proposed a relationship between the equivalent viscous damping ratio and ductility following the expression below:

$$\xi_{eq} = \xi_0 + a\left(1 - \frac{1}{\mu^b}\right) \tag{4.38}$$

Where `a` and `b` are constants that vary between 20 and 30, and 0.5 and 1, respectively, depending on the hysteretic properties of the structure. Thus, this model can be employed for various structure types, by adjusting these two constants. Give nthe fact that most of the current damping models have been derived or calibrated for reinforced concrete structures, a decision was made to adjust these parameters for the assessment of masonry structures. The `a` and `b` constants have been set to 25 and 0.5, respectively, as proposed by **BorziEtAl2008a** Nonetheless, due to the open and transparent arquitecture of the RMTK, any user can modify these parameters.

In order to use this model, the variable `damping_model` must be set to `Calvi_1999`.

The performance point (or target displacement) calculated within this methodology is equivalent to what would be obtained by subjecting the equivalent single degree of freedom oscilator to a nonlinear time history analysis. Then, estimated target displacement can be used to allocate the structure in a damage state, based on a pre-established set of displacement thresholds. This process can be repeated several times considering other ground motion records, as well as structures (i.e. building class).

In order to use this methodology, it is necessary to load one or multiple capacity curves and a set of ground motion records, as explained in Section 4.2.1 and 4.2.2, respectively. Then, it is necessary to specify a damage model using the parameter `damage_model` (see Section 4.2.3), and a damping ratio using the parameter `damping`. After importing the module `capacitySpectrumMethod`, it is possible to calculate the distribution of structures across the set of damage states for each ground motion record using the following command:

```
PDM, Sds = capacitySpectrumMethod.calculate_fragility(capacity_curves,...
gmrs,damage_model,damping)
```

Where `PDM` (i.e. probability damage matrix) represents a matrix with the number of structures in each damage state per ground motion record, and `Sds` (i.e. spectral displacements) represents a matrix with the maximum displacement (of the equivalent SDOF) of each structure per ground motion record. the variable PDM can then be used to calculate the mean fragility model as described in Section 4.7.1.

4.6.6 DBELA (Silva et al. 2013)

The Displacement-based Earthquake Loss Assessment (DBELA) methodology builds upon the urban assessment methodology proposed by **Calvi1999** in which the principles of structural mechanics and seismic response of buildings are used to estimate the seismic vulnerability of classes of buildings. The current implementation of the RMTK is only compatible with bare or infilled frame reinforced concrete structures.

In this method, the displacement capacity and demand for a number of limit states needs to be calculated. Each limit state marks the threshold between the levels of damage that a building might withstand, usually described by a reduction in strength or by exceedance of certain displacement/drift levels. Once these parameters are obtained, the displacement capacity of the first limit state is compared with the respective demand. If the demand exceeds the capacity, the next limit states need to be checked successively, until the demand no longer exceeds the capacity and the building damage state can be defined. If the demand also exceeds the capacity of the last limit state, the building is assumed to have collapsed. This procedure is schematically depicted in Figure 4.6, in which the capacities for three limit states are represented by Δ_i and the associated demand by Sd_i. In this example, the demand exceeds the capacity in the first and second limit state but not in the third limit state, thus allocating the building to the third damage state.

The calculation of the displacement capacity at each limit state is explained in the Model Generation Section (4.3), as this methodology can be employed to generate large sets of capacity curves (Sa versus Sd), which can be combined with other methodologies besides DBELA to derive fragility functions. Instead, this section is focused on describing how the seismic demand is handled in this methodology.

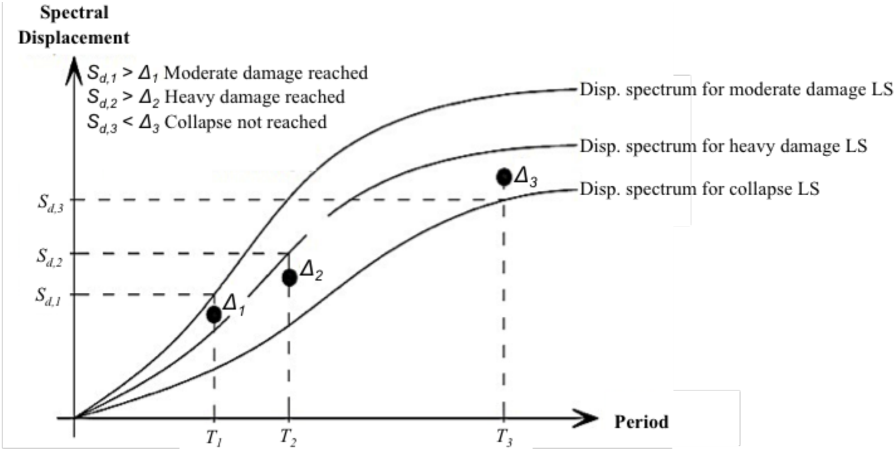

Figure 4.6 – *Comparison between limit state capacity and the associated demand (adapted from* **BalEtAl2010***).*

The demand is represented by a displacement spectrum which can be described as the expected displacement induced by an earthquake on a single-degree-of-freedom (SDOF) oscillator with a given period of vibration and viscous damping. This demand is initially calculated for a 5% viscous damping, and later modified for each limit state using a correction factor (η), representative of the equivalent viscous damping and ductility at the associated damage state. In the Eurocode 8 (**CEN2005**), the following equation is proposed for the calculation of the correction factor:

$$\eta_{LS_i} = \sqrt{\frac{10}{5+\xi_{eq_i}}} \tag{4.39}$$

Where ξ_{eq_i} stands for the equivalent viscous damping at the limit state i. Although in theory there is a multitude of damping models in the literature that could be used to calculate this equivalent viscous damping (see Section 4.6.5 for a description of the damping models implemented within the Capacity Spectrum Method), this method has been tested following the proposals by **PriestleyEtAl2007** for reinforced concrete frames (e.g. **BalEtAl2010 SilvaEtAl2013**). This model uses the following equation:

$$\xi_{eq} = 0.05 + 0.565\left(\frac{\mu - 1}{\pi\mu}\right) \tag{4.40}$$

Where μ_i stands for the ductility at the limit state i (assumed as the ratio between Δ_i and Δ_y). More accurate approaches have recently been proposed to estimate the correction factors (η), considering additional parameters, such as the magnitude or source-to-site distance (**RezaeianEtAl2012**).

With regards to the calculation of the yielding period (T_y) for bare frame structures, **CrowleyPinho2004** and **CrowleyEtAl2008** proposed a relationship between the period and the total height (H_T) of $0.10H_T$ and $0.07H_T$ for structures without and with lateral load design, respectively. For infilled frames, a relation equal to $0.06H_T$ has been recommended by **CrowleyPinho2006** for structures without lateral load design. The elongated period of vibration for any of the limit states (T_{LS_i})

can be computed using the following formula:

$$T_{LS_i} = T_y \sqrt{\frac{\mu_i}{1 + \alpha\mu_i - \alpha}} \tag{4.41}$$

where α stands for the post-yield stiffness ratio. In cases where this ratio can be assumed as zero, the relation between T_{LS_i} and T_y will depend purely on the limit state ductility as follows:

$$T_{LS_i} = T_y \sqrt{\mu_i} \tag{4.42}$$

In order to use this methodology, it is necessary to first assess the capacity displacement of one or multiple assets, following the DBELA approch explained in Section 4.3.1 (Model Generator). Moreover, a set of ground motion records and a damage model should be provided, as explained in Section 4.2.2 and 4.2.3), respectively. The type of structures that are being evaluated should be specified using the parameter `structure_type`. Currently this module of the RMTK accepts the options `bare frame` and `infilled frame`. After importing the module `DBELA`, it is possible to calculate the distribution of structures across the set of damage states for each ground motion record using the following command:

```
PDM, Sds = DBELA.calculate_fragility(capacity_curves,gmrs,...
damage_model,structure_type)
```

Where `PDM` (i.e. probability damage matrix) represents a matrix with the number of structures in each damage state per ground motion record, and `Sds` (i.e. spectral displacements) represents a matrix with the maximum displacement (of the equivalent SDOF) of each structure per ground motion record. the variable PDM can then be used to calculate the mean fragility model as described in Section 4.7.1.

4.6.7 Nonlinear time-history analysis in Single Degree of Freedom (SDOF) Oscilators

This methodology performs a series of non-linear time history analyses (NLTHA) over one or multiple single degree of freedom (SDOF) systems. In order to determine the structural capacity of the system(s) under analysis, it is necessary to identify the relationship between the base shear and roof displacement (i.e. pushover curve). This curve can be further modified in order to obtain the curve corresponding to an equivalent SDOF, or capacity curve. It is typically assumed that the fundamental mode of vibration corresponds with the predominant response of the structure. Under this hypothesis, the capacity curve usually represents the first mode of response of the structure. This is usually valid for buildings with fundamental periods of vibration up to approximately 1.0 s. Otherwise, higher modes should be taken into account. Along with the capacity curve, it is necessary to specify either the mass or the fundamental period of vibration of the SDOF system.

On the other hand, the demand is represented by a set of ground motion records. The response of each structure is given by the solution of the equation of motion for an inelastic SDOF under earthquake excitation:

$$m\ddot{u}(t) + c\dot{u}(t) + ku(t) = p(t) \tag{4.43}$$

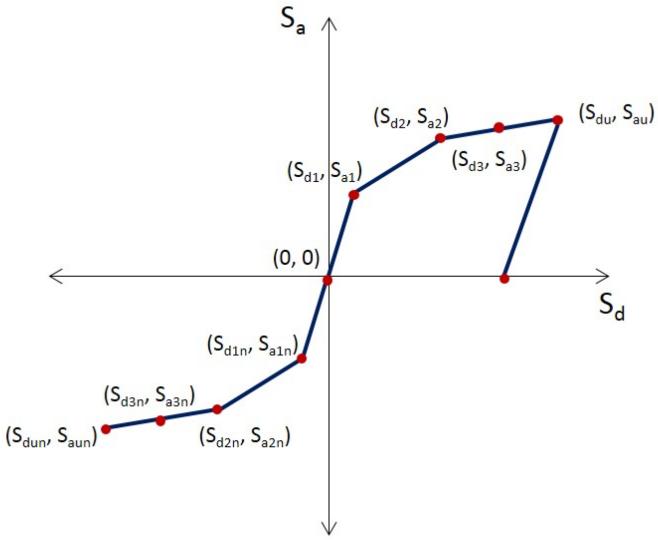

Figure 4.7 – *Representation of the capacity curve required to represent the structural capacity of the SDOF system.*

Where u, $\dot{u}$ and $\ddot{u}$ stand for the displacement, velocity and acceleration, respectively, over time (t), and p represents an external excitation. In this method, the behaviour of the SDOF is modelled by a capacity curve that must follow the structure ilustrated in Figure 4.7.

Six relevant points should be defined in this curve. The first one corresponds to the origin, (0, 0). The next point, (Sdy, Say), corresponds to the yielding point of the structure, i.e. the point beyond which the structure no longer displays an elastic behaviour. The following two points are defined as any two intermediate points between the yield point and the ultimate point which can be used to represent particular structural properties, such as reduction of stifness due to collapse of infill pannels or softning behavior due to P-delta effects. The ultimate point (Sdu, Sau) corresponds to the point of maximum displacement of the structure. Finally, the last point (Sdu, 0) is just a control point to define the hysteretic model, in case degradation is considered in the analysis. For the purposes of the RMTK, the first five points must be provided as input. The nonlinear time history analysis are performed using the open-source software for structural analysis OpenSees (**McKennaEtAl2000**). It is important to understand the GEM Foundation does not have authorization to distriute this tool, and therefore in order to use this methodology, users are expected to download OpenSEES (http://opensees.berkeley.edu/), and allocate it in the appropriate folder (`vulnerability/derivation_fragility/NLTHA_on_SDOF`).

In order to use this methodology, it is necessary to load one or multiple capacity curves and a set of ground motion records, as explained in Section 4.2.1 and 4.2.2, respectively. Then, it is necessary to specify a damage model using the parameter `damage_model` (see Section 4.2.3). The damping ratio must be defined using the parameter `damping`, and if structural degradation should be considered in the analysis, it is necessary to set the parameter `degradation` to `True`. After importing the module `NLTHA_on_SDOF`, it is possible to calculate the distribution of structures across the set of damage states for each ground motion record using the following command:

```
PDM, Sds = NLTHA_on_SDOF.calculate_fragility(capacity_curves,gmrs,...
damage_model,damping)
```

Where `PDM` (i.e. probability damage matrix) represents a matrix with the number of structures in each damage state per ground motion record, and `Sds` (i.e. spectral displacements) represents a matrix with the maximum displacement (of the equivalent SDOF) of each structure per ground motion record. the variable PDM can then be used to calculate the mean fragility model as described in Section 4.7.1.

4.7 Derivation of fragility and vulnerability functions

4.7.1 Derivation of fragility functions

To be completed by Anirudh.

4.7.2 Derivation of vulnerability functions

To be completed by Anirudh.

Part I

Appendices

A. The 10 Minute Guide to Python!

The RMTK is intended to be used by scientists and engineers without the necessity of having an existing knowledge of Python. It is hoped that the examples contained in this manual should provide enough context to allow the user to understand how to use the tools for their own needs. In spite of this, however, an understanding of the fundamentals of the Python programming language can greatly enhance the user experience and permit the user to join together the tools in a workflow that best matches their needs.

The aim of this appendix is therefore to introduce some fundamentals of the Python programming language in order to help understand how, and why, the RMTK can be used in a specific manner. If the reader wishes to develop their knowledge of the Python programming language beyond the examples shown here, there is a considerable body of literature on the topic from both a scientific and developer perspective.

A.1 Basic Data Types

Fundamental to the use of the RMTK is an understanding of the basic data types Python recognises:

A.1.1 Scalar Parameters

- **float** A floating point (decimal) number. If the user wishes to enter in a floating point value then a decimal point must be included, even if the number is rounded to an integer.

```
>> a = 3.5
>> print a, type(a)
3.5 <type 'float'>
```

- **integer** An integer number. If the decimal point is omitted for a floating point number the number will be considered an integer

```
>> b = 3
>> print b, type(b)
3 <type 'int'>
```

The functions `float()` and `int()` can convert an integer to a float and vice-versa. Note that taking `int()` of a fraction will round the fraction down to the nearest integer

```
>> float(b)
3
>> int(a)
3
```

- **string** A text string (technically a “list” of text characters). The string is indicated by the quotation marks ”something” or ’something else’

```
>> c = "apples"
>> print c, type(c)
apples <type 'str'>
```

- **bool** For logical operations python can recognise a variable with a boolean data type (`True` / `False`).

```
>> d = True
>> if d:
        print "y"
    else:
        print "n"
y
>> d = False
>> if d:
        print "y"
    else:
        print "n"
n
```

Care should be taken in Python as the value 0 and 0.0 are both recognised as False if applied to a logical operation. Similarly, booleans can be used in arithmetic where True and False take the values 1 and 0 respectively

```
>> d = 1.0
>> if d:
        print "y"
    else:
        print "n"
y
>> d = 0.0
>> if d:
        print "y"
    else:
        print "n"
n
```

A.1.1.1 Scalar Arithmetic

Scalars support basic mathematical operations (# indicates a comment):

```
>> a = 3.0
>> b = 4.0
>> a + b # Addition
7.0
>> a * b # Multiplication
12.0
>> a - b # Subtraction
-1.0
>> a / b # Division
0.75
>> a ** b  # Exponentiation
81.0
# But integer behaviour can be different!
>> a = 3; b = 4
```

```
>> a / b
0
>> b / a
1
```

A.1.2 Iterables

Python can also define variables as lists, tuples and sets. These data types can form the basis for iterable operations. It should be noted that unlike other languages, such as Matlab or Fortran, Python iterable locations are zero-ordered (i.e. the first location in a list has an index value of 0, rather than 1).

- **List** A simple list of objects, which have the same or different data types. Data in lists can be re-assigned or replaced

```
>> a_list = [3.0, 4.0, 5.0]
>> print a_list
[3.0, 4.0, 5.0]
>> another_list = [3.0, "apples", False]
>> print another_list
[3.0, 'apples', False]
>> a_list[2] = -1.0
a_list = [3.0, 4.0, -1.0]
```

- **Tuples** Collections of objects that can be iterated upon. As with lists, they can support mixed data types. However, objects in a tuple cannot be re-assigned or replaced.

```
>> a_tuple = (3.0, "apples", False)
>> print a_tuple
(3.0, 'apples', False)
# Try re-assigning a value in a tuple
>> a_tuple[2] = -1.0
TypeError                        Traceback (most recent call last)
<ipython-input-43-644687cfd23c> in <module>()
----> 1 a_tuple[2] = -1.0

TypeError: 'tuple' object does not support item assignment
```

- **Range** A range is a convenient function to generate arithmetic progressions. They are called with a `start`, a `stop` and (optionally) a `step` (which defaults to 1 if not specified)

```
>> a = range(0, 5)
>> print a
[0, 1, 2, 3, 4]   # Note that the stop number is not
                  # included in the set!
>> b = range(0, 6, 2)
>> print b
[0, 2, 4]
```

- **Sets** A set is a special case of an iterable in which the elements are unordered, but contains more enhanced mathematical set operations (such as intersection, union, difference, etc.)

```
>> from sets import Set
>> x = Set([3.0, 4.0, 5.0, 8.0])
>> y = Set([4.0, 7.0])
>> x.union(y)
Set([3.0, 4.0, 5.0, 7.0, 8.0])
>> x.intersection(y)
Set([4.0])
>> x.difference(y)
Set([8.0, 3.0, 5.0]) # Notice the results are not ordered!
```

A.1.2.1 Indexing

For some iterables (including lists, sets and strings) Python allows for subsets of the iterable to be selected and returned as a new iterable. The selection of elements within the set is done according to the `index` of the set.

```
>> x = range(0, 10)  # Create an iterable
>> print x
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>> print x[0] # Select the first element in the set
0            # recall that iterables are zero-ordered!
>> print x[-1] # Select the last element in the set
9
>> y = x[:] # Select all the elements in the set
>> print y
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>> y = x[:4]  # Select the first four element of the set
>> print y
[0, 1, 2, 3]
>> y = x[-3:] # Select the last three elements of the set
>> print y
[7, 8, 9]
>> y = x[4:7] # Select the 4th, 5th and 6th elements
>> print y
[4, 5, 6]
```

A.1.3 Dictionaries

Python is capable of storing multiple data types associated with a map of variable names inside a single object. This is called a “Dictionary”, and works in a similar manner to a “data structure” in languages such as Matlab. Dictionaries are used frequently in the RMTK as ways of structuring inputs to functions that share a common behaviour but may take different numbers and types of parameters on input.

```
>> earthquake = {"Name": "Parkfield",
                 "Year": 2004,
                 "Magnitude": 6.1,
                 "Recording Agencies" = ["USGS", "ISC"]}
# To call or view a particular element in a dictionary
>> print earthquake["Name"], earthquake["Magnitude"]
Parkfield 6.1
```

A.1.4 Loops and Logicals

Python’s syntax for undertaking logical operations and iterable operations is relatively straightforward.

A.1.4.1 Logical

A simple logical branching structure can be defined as follows:

```
>> a = 3.5
>> if a <= 1.0:
       b = a + 2.0
   elif a > 2.0:
       b = a - 1.0
   else:
       b = a ** 2.0
>> print b
2.5
```

Boolean operations can are simply rendered as `and`, `or` and `not`.

```
>> a = 3.5
>> if (a <= 1.0) or (a > 3.0):
        b = a - 1.0
   else:
        b = a ** 2.0
>> print b
2.5
```

A.1.4.2 Looping

There are several ways to apply looping in python. For simple mathematical operations, the simplest way is to make use of the **range** function:

```
>> for i in range(0, 5):
        print i, i ** 2
0   0
1   1
2   4
3   9
4   16
```

The same could be achieved using the `while` function (though possibly this approach is far less desirable depending on the circumstance):

```
>> i = 0
>> while i < 5:
        print i, i ** 2
        i += 1
0   0
1   1
2   4
3   9
4   16
```

A `for` loop can be applied to any iterable:

```
>> fruit_data = ["apples", "oranges", "bananas", "lemons",
                 "cherries"]
>> i = 0
>> for fruit in fruit_data:
        print i, fruit
        i += 1
0   apples
1   oranges
2   bananas
3   lemons
4   cherries
```

The same results can be generated, arguably more cleanly, by making use of the `enumerate` function:

```
>> fruit_data = ["apples", "oranges", "bananas", "lemons",
                 "cherries"]
>> for i, fruit in enumerate(fruit_data):
        print i, fruit
0   apples
1   oranges
2   bananas
3   lemons
4   cherries
```

As with many other programming languages, Python contains the statements `break` to break out of a loop, and `continue` to pass to the next iteration.

```
>> i = 0
>> while i < 10:
        if i == 3:
            i += 1
            continue
        elif i == 5:
            break
        else:
            print i, i ** 2
        i += 1
0  0
1  1
2  4
4  16
```

A.2 Functions

Python easily supports the definition of functions. A simple example is shown below. *Pay careful attention to indentation and syntax!*

```
>> def a_simple_multiplier(a, b):
        """
        Documentation string - tells the reader the function
        will multiply two numbers, and return the result and
        the square of the result
        """
        c = a * b
        return c, c ** 2.0

>> x = a_simple_multiplier(3.0, 4.0)
>> print x
(12.0, 144.0)
```

In the above example the function returns two outputs. If only one output is assigned then that output will take the form of a tuple, where the elements correspond to each of the two outputs. To assign directly, simply do the following:

```
>> x, y = a_simple_multiplier(3.0, 4.0)
>> print x
12.0
>> print y
144.0
```

A.3 Classes and Inheritance

Python is one of many languages that is fully object-oriented, and the use (and terminology) of objects is prevalent throughout the RMTK and this manual. A full treatise on the topic of object oriented programming in Python is beyond the scope of this manual and the reader is referred to one of the many textbooks on Python for more examples

A.3.1 Simple Classes

A class is an object that can hold both attributes and methods. For example, imagine we wish to convert an earthquake magnitude from one scale to another; however, if the earthquake occurred after a user-defined year we wish to use a different formula. This could be done by a method, but we can also use a class:

```
>> class MagnitudeConverter(object):
       """
       Class to convert magnitudes from one scale to another
       """
       def __init__(self, converter_year):
           """
           """
           self.converter_year = converter_year

       def convert(self, magnitude, year):
           """
           Converts the magnitude from one scale to another
           """
           if year < self.converter_year:
               converted_magnitude = -0.3 + 1.2 * magnitude
           else:
               converted_magnitude = 0.1 + 0.94 * magnitude
           return converted_magnitude

>> converter1 = MagnitudeConverter(1990)
>> mag_1 = converter1.convert(5.0, 1987)
>> print mag_1
5.7
>> mag_2 = converter1.convert(5.0, 1994)
>> print mag_2
4.8
# Now change the conversion year
>> converter2 = MagnitudeConverter(1995)
>> mag_1 = converter2.convert(5.0, 1987)
>> print mag_1
5.7
>> mag_2 = converter2.convert(5.0, 1994)
>> print mag_2
5.7
```

In this example the class holds both the attribute `converter_year` and the method to convert the magnitude. The class is created (or "instantiated") with only the information regarding the cut-off year to use the different conversion formulae. Then the class has a method to convert a specific magnitude depending on its year.

A.3.2 Inheritance

Classes can be useful in many ways in programming. One such way is due to the property of inheritance. This allows for classes to be created that can inherit the attributes and methods of another class, but permit the user to add on new attributes and/or modify methods.

In the following example we create a new magnitude converter, which may work in the same way as the `MagnitudeConverter` class, but with different conversion methods.

```
>> class NewMagnitudeConverter(MagnitudeConverter):
       """
       A magnitude converter using different conversion
       formulae
       """
       def convert(self, magnitude, year):
           """
           Converts the magnitude from one scale to another
           - differently!!!
           """
           if year < self.converter_year:
               converted_magnitude = -0.1 + 1.05 * magnitude
```

```
            else:
                converted_magnitude = 0.4 + 0.8 * magnitude
            return converted_magnitude
# Now compare converters
>> converter1 = MagnitudeConverter(1990)
>> converter2 = NewMagnitudeConverter(1990)
>> mag1 = converter1.convert(5.0, 1987)
>> print mag1
5.7
>> mag2 = converter2.convert(5.0, 1987)
>> print mag2
5.15
>> mag3 = converter1.convert(5.0, 1994)
>> print mag3
4.8
>> mag4 = converter2.convert(5.0, 1994)
>> print mag4
4.4
```

A.3.3 Abstraction

Inspection of the RMTK code (https://github.com/GEMScienceTools/rmtk) shows frequent usage of classes and inheritance. This is useful in our case if we wish to make available different methods for the same problem. In many cases the methods may have similar logic, or may provide the same types of outputs, but the specifics of the implementation may differ. Functions or attributes that are common to all methods can be placed in a “Base Class”, permitting each implementation of a new method to inherit the “Base Class” and its functions/attributes/behaviour. The new method will simply modify those aspects of the base class that are required for the specific method in question. This allows functions to be used interchangeably, thus allowing for a "mapping" of data to specific methods.

An example of abstraction is shown using our two magnitude converters shown previously. Imagine that a seismic recording network (named "XXX") has a model for converting from their locally recorded magnitude to a reference global scale (for the purposes of narrative, imagine that a change in recording procedures in 1990 results in a change of conversion model). A different recording network (named “YYY”) has a different model for converting their local magnitude to a reference global scale (and we imagine they also changed their recording procedures, but they did so in 1994). We can create a mapping that would apply the correct conversion for each locally recorded magnitude in a short catalogue, provided we know the local magnitude, the year and the recording network.

```
>> CONVERSION_MAP = {"XXX": MagnitudeConverter(1990),
                     "YYY": NewMagnitudeConverter(1994)}
>> earthquake_catalogue = [(5.0, "XXX", 1985),
                           (5.6, "YYY", 1992),
                           (4.8, "XXX", 1993),
                           (4.4, "YYY", 1997)]
>> for earthquake in earthquake_catalogue:
       converted_magnitude = \ # Line break for long lines!
           CONVERSION_MAP[earthquake[1]].convert(earthquake[0],
                                                 earthquake[2])
       print earthquake, converted_magnitude
(5.0, "XXX", 1985) 5.7
(5.6, "YYY", 1992) 5.78
(4.8, "XXX", 1993) 4.612
(4.4, "YYY", 1997) 3.92
```

So we have a simple magnitude homogenisor that applies the correct function depending on the network and year. It then becomes a very simple matter to add on new converters for new agencies; hence we have a "toolkit" of conversion functions!

A.4 Numpy/Scipy

Python has two powerful libraries for undertaking mathematical and scientific calculation, which are essential for the vast majority of scientific applications of Python: Numpy (for multi-dimensional array calculations) and Scipy (an extensive library of applications for maths, science and engineering). Both libraries are critical to both OpenQuake and the RMTK. Each package is so extensive that a comprehensive description requires a book in itself. Fortunately there is abundant documentation via the online help for Numpy www.numpy.org and Scipy www.scipy.org, so we do not need to go into detail here.

The particular facet we focus upon is the way in which Numpy operates with respect to vector arithmatic. Users familiar with Matlab will recognise many similarities in the way the Numpy package undertakes array-based calculations. Likewise, as with Matlab, code that is well vectorised is signficantly faster and more efficient than the pure Python equivalent.

The following shows how to undertake basic array arithmetic operations using the Numpy library

```
>> import numpy as np
# Create two vectors of data, of equal length
>> x = np.array([3.0, 6.0, 12.0, 20.0])
>> y = np.array([1.0, 2.0, 3.0, 4.0])
# Basic arithmetic
>> x + y    # Addition (element-wise)
np.array([4.0, 8.0, 15.0, 24.0])
>> x + 2    # Addition of scalar
np.array([5.0, 8.0, 14.0, 22.0])
>> x * y    # Multiplication (element-wise)
np.array([3.0, 12.0, 36.0, 80.0])
>> x * 3.0    # Multiplication by scalar
np.array([9.0, 18.0, 36.0, 60.0])
>> x - y    # Subtraction (element-wise)
np.array([2.0, 4.0, 9.0, 16.0])
>> x - 1.0    # Subtraction of scalar
np.array([2.0, 5.0, 11.0, 19.0])
>> x / y    # Division (element-wise)
np.array([3.0, 3.0, 4.0, 5.0])
>> x / 2.0    # Division over scalar
np.array([1.5, 3.0, 6.0, 10.0])
>> x ** y    # Exponentiation (element-wise)
np.array([3.0, 36.0, 1728.0, 160000.0])
>> x ** 2.0    # Exponentiation (by scalar)
np.array([9.0, 36.0, 144.0, 400.0])
```

Numpy contains a vast set of mathematical functions that can be operated on a vector (e.g.):

```
>> x = np.array([3.0, 6.0, 12.0, 20.0])
>> np.exp(x)
np.array([2.00855369e+01, 4.03428793e+02, 1.62754791e+05,
          4.85165195e+08])
# Trigonometry
>> theta = np.array([0., np.pi / 2.0, np.pi, 1.5 * np.pi])
>> np.sin(theta)
np.array([0.0000, 1.0000, 0.0000, -1.0000])
>> np.cos(theta)
np.array([1.0000, 0.0000, -1.0000, 0.0000])
```

Some of the most powerful functions of Numpy, however, come from its logical indexing:

```
>> x = np.array([3.0, 5.0, 12.0, 21.0, 43.0])
>> idx = x >= 10.0    # Perform a logical operation
>> print idx
np.array([False, False, True, True, True])
>> x[idx]    # Return an array consisting of elements
             # for which the logical operation returned True
np.array([12.0, 21.0, 43.0])
```

Create, index and slice n-dimensional arrays:

```
>> x = np.array([[3.0,  5.0, 12.0, 21.0, 43.0],
                 [2.0,  1.0,  4.0, 12.0, 30.0],
                 [1.0, -4.0, -2.1,  0.0, 92.0]])
>> np.shape(x)
(3, 5)
>> x[:, 0]
np.array([3.0, 2.0, 1.0])
>> x[1, :]
np.array([2.0, 1.0, 4.0, 12.0, 30.0])
>> x[:, [1, 4]]
np.array([[ 5.0, 43.0],
          [ 1.0, 30.0],
          [-4.0, 92.0]])
```

The reader is referred to the online documentation for the full set of functions!

B. Bibliography