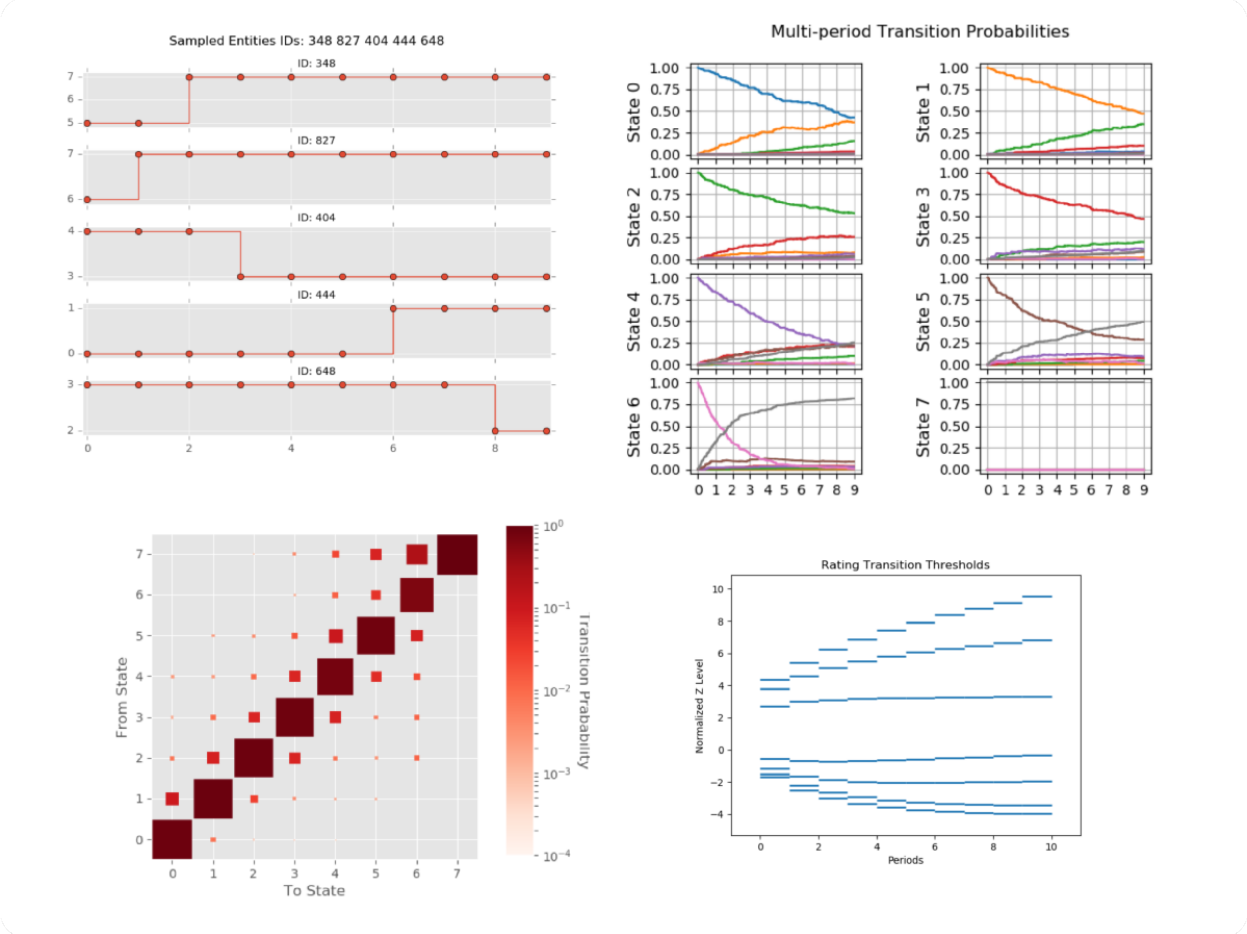
transitionMatrix Documentation

Release 0.5.0

Open Risk

Feb 22, 2023

Contents:



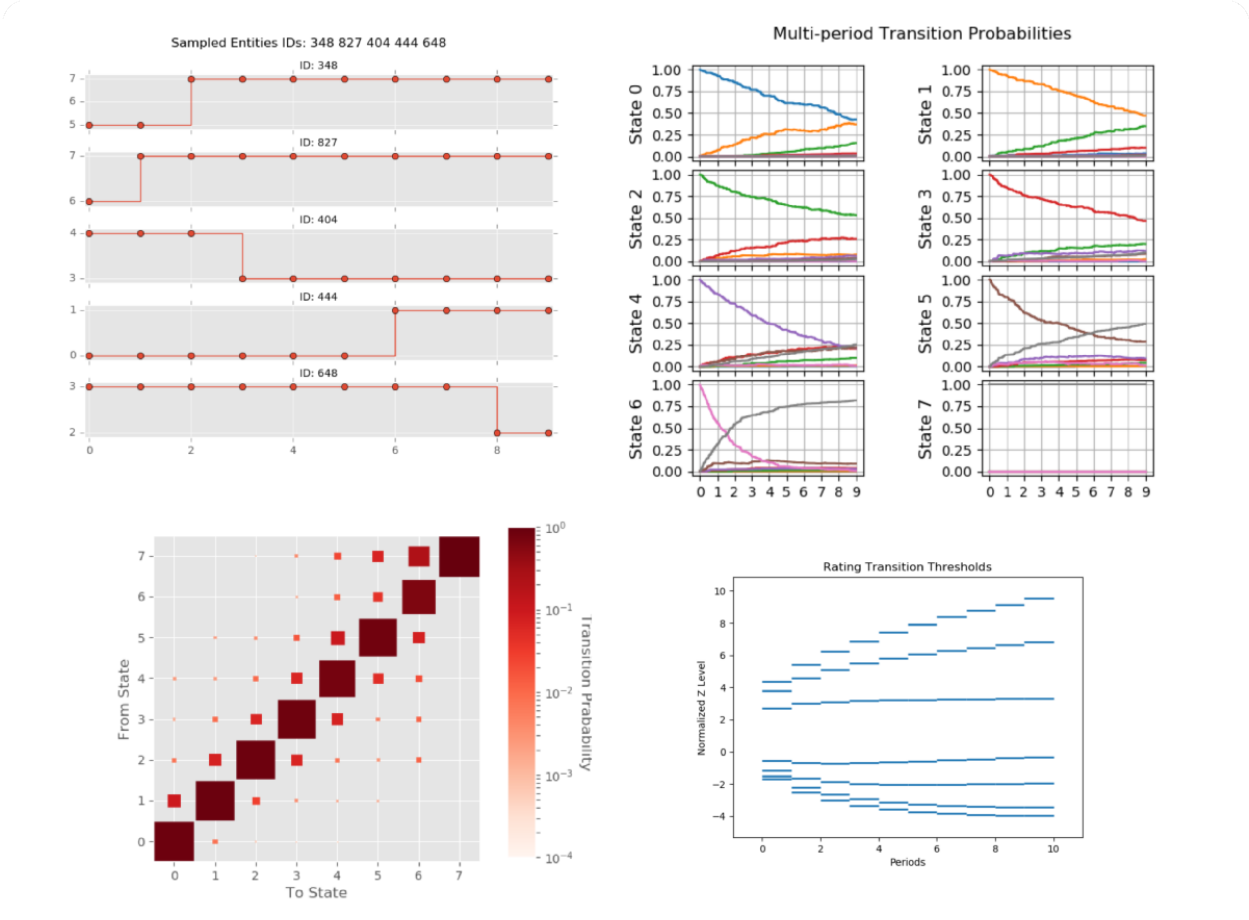
transitionMatrix is a pure Python powered library for the statistical analysis and visualization of state transition phenomena. It can be used to analyze any dataset that captures *timestamped transitions in a discrete state space.*

Use cases include applications in finance (for example credit rating transitions), IT (system state event logs) and more.

NB: transitionMatrix is still in alpha release / active development. If you encounter issues please raise them in our github repository

CHAPTER 1

The transitionMatrix Library



transitionMatrix is a pure Python powered library for the statistical analysis and visualization of state transition phenomena. It can be used to analyze any dataset that captures *timestamped transitions in a discrete state space.*

- Author: Open Risk
- License: Apache 2.0
- Development Website: Github
- Code Documentation: Read The Docs
- Mathematical Documentation: Open Risk Manual
- Chat: Open Risk Commons
- Training: Open Risk Academy
- Showcase: Blog Posts

1.1 Functionality

You can use transitionMatrix to:

- **Estimate** transition matrices from historical event data using a variety of estimators
- **Characterise** transition matrices (identify their key properties)
- **Visualize** event data and transition matrices
- **Manipulate** transition matrices (derive generators, perform comparisons, stress transition rates etc.)
- Access standardized Datasets for testing
- Extract and work with credit default curves (absorbing states)
- Map credit ratings using mapping tables
- More (still to be documented :-)

1.2 Architecture

- transitionMatrix provides intuitive objects for handling transition matrices individually and as sets (based on numpy arrays)
- supports file input/output in json and csv formats
- it has a powerful API for handling event data (based on pandas and numpy)
- supports visualization using matplotlib

CHAPTER 2

Installation

You can install and use the transitionMatrix package in any system that supports the Scipy ecosystem of tools

2.1 Dependencies

- TransitionMatrix requires Python 3 (currently 3.7)
- It depends on numerical and data processing Python libraries (Numpy, Scipy, Pandas).
- The Visualization API depends on Matplotlib.
- The precise dependencies are listed in the requirements.txt file.
- TransitionMatrix may work with earlier versions of python / these packages but it is not tested.

2.2 From PyPI

```
pip3 install transitionMatrix
```

2.3 From sources

Download the sources in your preferred directory:

```
git clone https://github.com/open-risk/transitionMatrix
```

2.4 Using virtualenv

It is advisable to install the package in a virtualenv so as not to interfere with your system's python distribution

```
virtualenv -p python3 tm_test
source tm_test/bin/activate
```

If you do not have pandas already installed make sure you install it first (this will also install numpy and other required dependencies).

```
pip3 install -r requirements.txt
```

Finally issue the install command and you are ready to go!

```
python3 setup.py install
```

2.5 File structure

The distribution has the following structure:

```
| transitionMatrix/     Directory with the library source code
| -- model.py           File with main data structures
| -- estimators/        Directory with the estimator methods
| -- statespaces/       Directory with state space objects and methods
| -- creditratings/     Directory with predefined credit rating structures
| -- generators/        Directory with data generator methods
| -- utils/             Directory with helper classes and methods
| -- examples/          Directory with usage examples
| ---- python/          Examples as standalone python scripts
| ---- notebooks/       Examples as jupyter notebooks
| -- datasets/          Directory with a variety of datasets useful for getting␣
↪started
| -- tests/             Directory with the testing suite
```

2.6 Other similar open source software

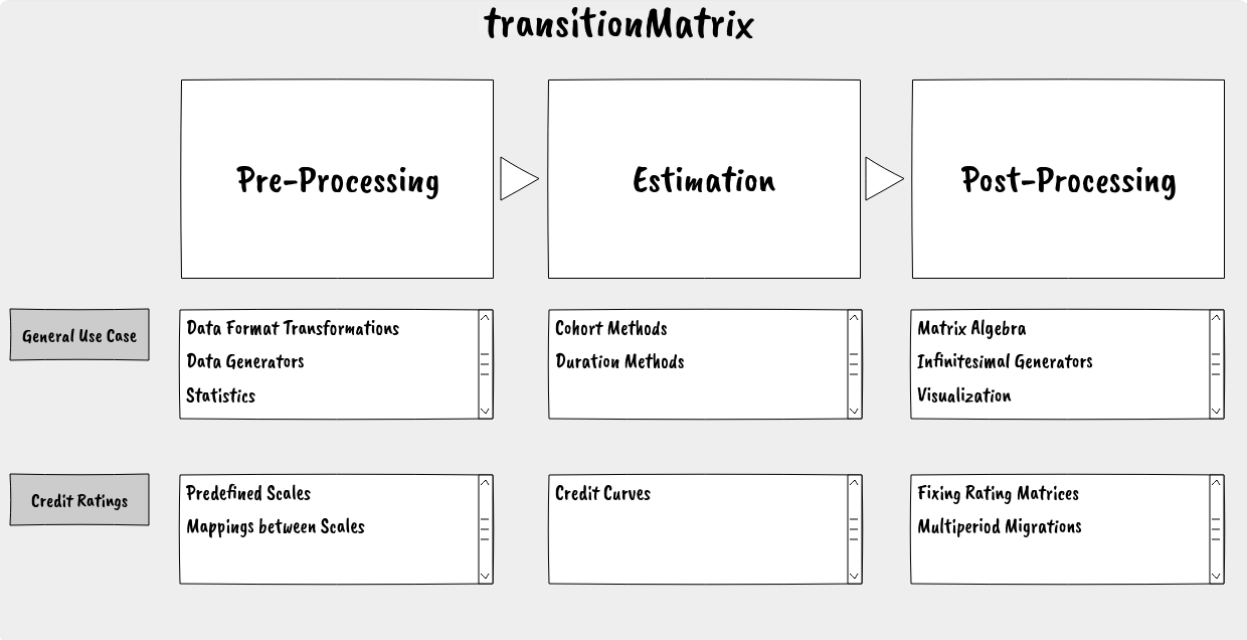
- etm, an R package for estimating empirical transition matrices
- msSurv, an R Package for Nonparametric Estimation of Multistate Models
- msm, Multi-state modelling with R
- mstate, competing risks and multistate models in R
- lifelines, python survival package

CHAPTER 3

Getting Started

3.1 The transitionMatrix components

The transitionMatrix package includes several components (organized in sub-packages) providing a variety of functionality for working with state transition phenomena. The overall organization and functionality is summarized in the following graphic:



The library is structured in a modular way: users may mix and match the various components to meet their own needs. The main workflow can be captured in the standard pre-processing, modelling and post-processing stages:

- *Preprocessing* stage
- *Estimation* stage

- *Post-processing* stage

An secondary segmentation that is important to keep in mind is between general functionality that is relevant to general data about state transitions and more specific domains with more specific conventions and needs. At present the only specific domain concerns credit ratings.

Here we will dive straight-in into using transitionMatrix going a concrete (and typical) example using historical credit rating transitions. Further resources and links to more detailed and specific usage are available at the end of this section. People who are not at all familiar with the machinery of transition matrices might want to start with *Basic Operations*.

3.2 An end-to-end usage example from credit risk

In order to give a quick introduction to the package we discuss here a concrete and end-to-end example of using transitionMatrix that is drawn for the credit ratings space. The example does not cover all functionality, but it demonstrates the core workflow.

We will use the data set "rating_data.csv" that is available in *Datasets* directory. The code snippets discussed here are all from the script <examples/python/estimate_matrix.py>

3.2.1 Step 1: Loading the data

Data loading is best done via pandas dataframes:

```
data = pd.read_csv('../../datasets/rating_data.csv')
print(data.head())

   CustomerId        Date Rating  RatingNum
0           1  30-05-2000   CCC+          7
1           1  31-12-2000     B+          6
2           2  21-05-2003     B+          6
3           3  30-12-1999    BB+          5
4           3  30-10-2000     B+          6
```

We see that there is just enough metadata in the csv header to get an impression of how the data set captures transitions:

- Each entity is identified by an integer (First column: CustomerId)
- State measurements / transitions are observed at dates (in the DD-MM-YYYY format) (Second column: Date)
- There is an implied credit rating scale using symbols 'B+, BB+' etc (Third column: Rating)
- The rating scale is also expressed as integers (Fourth column: RatingNum)

Note: There are several important points we need to clarify before we can confidently extract information from this dataset and (ultimately) estimate a transition matrix. For example:

- Do we understand the column labels or do we need additional (metadata)
- What is the observation window?
- Are the dates indicating a measurement (including no change) or a changed state?
- Are all possible transitions observed in the sample?
- Are all data provided consistent?
- Etc

Some of those questions maybe answerable with the tools offered by transitionMatrix but it is always the responsibility of the data scientist to make sure they are correctly interpreting the data and using the tools accordingly! The

3.2.2 Step 2: Understanding the data format

Our first task is to identify which data format is closest for us to use. In *Input Data Formats* we see that what we have *looks* closest to a *Compact Form of Long Format* with the temporal information in *String Dates*.

3.2.3 Step 3: Data cleaning and normalization

Having data in the right format is only the first step!

Warning: As mentioned above, we need to be careful that the input data are "clean" and have unambiguous interpretation. Here are some examples of potential issues:

Example 1: The entity with ID=41 has only one measurement and it is NR. What does it mean? Can we remove it from the data without impact?

- 40, 30-12-2003, A+, 3
- 41, 21-07-2000, NR, 0
- 42, 30-06-2004, A+, 3

Example 2: ID 46 has three identical measurements at different times. What does it mean? Can we ignore the intermediate observations without impact? (Observing a no-change is no the same as not observing a change!)

- 46, 30-05-1999, AA+, 2
- 46, 30-08-2001, AA+, 2
- 46, 30-12-2002, AA+, 2
- 46, 30-12-2003, A+, 3

Example 3: ID 54 is transitioning to D (absorbing state) and then to NR. This means that the label 'NR' is used in multiple ways: Something that is not rated because we know its state anyway (D) and something that

- 54,30-10-2001, CCC+, 7
- 54,30-07-2002, D, 8
- 54,30-12-2002, NR, 0

Those examples illustrate that converting the raw input data into a clean dataset might require additional assumptions. This must be done on a case-by-case basis. For example: if an entity is only observed once in a state, maybe it is valid to assume it is in that state throughout the observation window. Another example: maybe it is valid to assume that multiple observations of no changing state do not carry any information and thus can be merged, etc.

Note: For a (non-exhaustive) list of data cleaning steps check out the script examples/python/data_cleaning_example.py

3.2.4 Step 4: Establish the State Space

Lets rename the columns accordingly:

```
data = data.rename(columns={"Rating": "State", "Date": "Time", "CustomerId": "ID"})
print(unique_states(data))

['CCC+' 'B+' 'BB+' 'AA+' 'A+' 'BBB+' 'NR' 'D' 'AAA']
```

We see that we have 9 unique states:

- 7 ratings states: AAA, AA+, etc presumably refer to different credit qualities (it is typical when the rating scale uses the (+) qualifier to also have (-) but here this is not the case).
- D probably means an *absorbing* (Default) state
- NR probably means *not rated*

Let us create the State Space

```
originator = 'me'
full_name = 'my state space'
definition = [('0', 'NR'), ('1', "AAA"), ('2', "AA+"), ('3', "A+"), ('4', "BBB+"),
              ('5', "BB+"), ('6', "B+"), ('7', "CCC+"),
              ('8', "D")]

mySS = StateSpace(definition=definition, originator=originator, full_name=full_name,␣
↪cqs_mapping=None)

print(mySS.validate_dataset(data))
```

Note: The above shows the functionality of the StateSpace object. In this case the validation is expected as we constructed the labels from what we found on the data set, but if the rating scale we use is given this becomes a more insightful validation exercise

3.3 Further Resources

There is a large and growing set of examples and other training material to get you started:

3.3.1 Examples Directory

Look at the *Usage Examples* directory of the transitionMatrix distribution for a variety of typical workflows.

Note: Many scripts contain *multiple examples*. You need to manually edit the example ID within the file to select the desired example

3.3.2 Open Risk Academy

For more in depth study, the Open Risk Academy has courses elaborating on the use of the library:

- Analysis of Credit Migration using Python TransitionMatrix

Note: The Example scripts from the Open Risk Academy course PYT26038 are available in a separate repo

CHAPTER 4

Input Data Formats

The transitionMatrix package supports a variety of input data formats for empirical (observation) data. Two key ones are described here in more detail. More background about data formats is available at the Open Risk Manual Risk Data Category

4.1 Long Data Format

Long Data Format is a tabular representation of time series data that records the states (measurements) of multiple entities. Its defining characteristic is that each table row contains data pertaining to one entity at one point in time.

4.1.1 Canonical Form of Long Data

The Long Data Format (also Narrow or Stacked) consists of Tuples, e.g. (Entity ID, Time, From State, To State) indicating the time T at which an entity with ID migrated from the (From State) -> to the (To State).

The *canonical form* used as input to duration based estimators uses normalized timestamps (from 0 to T_max, where T_max is the last timepoint) and looks as follows:

ID	Time	From	To
1	1.1	0	1
1	2.0	1	2
1	3.4	2	3
1	4.0	3	2
2	1.2	0	1
2	2.4	1	2
2	3.5	2	3

The canonical form has the advantage of being unambiguous about the context where the transition occurs. The meaning of each row of data stands on its own and does not rely on the order (or even the presence) of other records.

This facilitates, for example, the algorithmic processing of the data. On the flipside, the format is less efficient in terms of storage (the state information occurs twice) compared to the compact format (See below).

The canonical format requires that the final state of all entities at the end of the observation window (Time F) is included (otherwise we have no indication about when the measurements stopped). Alternatively such information is provided as separate metadata (or implicitly, for example if measurements are understood to span a number of full annual periods).

Note: Synthetic_data(7, 8, 9) in the *Datasets* collection are examples of data in long format and canonical form

---

4.1.2 String Dates

It is frequent that transition data (e.g. from financial applications) have timestamps in the form of a *date string*. For example:

ID	Date String	From	To
1	10-10-2010	0	1
1	10-11-2010	1	2

String dates must be converted to a numerical representation before we can work with the transition data. transitionMatrix offers the `transitionMatrix.utils.converters.datetime_to_float()` function of `transitionMatrix.utils` subpackage can be used to convert data into the canonical form.

Note: Synthetic_data9 and rating_data in the *Datasets* collection have observation times in string data form.

4.2 Compact Form of Long Format

The format uses triples (ID, Time, State), indicating the time T at which an entity ID **Left** its previous state S (the state it migrates to is encoded in the next observation of the same entity). The convention can obviously be reversed to indicate the time of entering a new state (in which case we need some information to bound the start of the observation window).

The compact long format avoids the duplication of data of the canonical approach but requires the presence of other records to infer the realised sequence of events.

The format also requires that the final state of all entities at the end of the observation window (Time F) is included as the last record (otherwise we have no indication about when the measurements stopped). Alternatively such information is provided separately (or implicitly, e.g. if measurements are understood to span a number of full annual periods).

ID	Time	State
1	1.1	0
1	2.0	1
1	3.4	2
1	4.0	3
1	F	2
2	1.2	0
2	2.4	1
2	3.5	2
2	F	3

4.3 Wide Data Format

Wide Data Format is an alternative tabular representation of time series data that records the states (measurements) of multiple entities. Its defining characteristic is that each table row contains *all the data* pertaining to any one entity. The measurement times are not arbitrary but encoded in the column labels:

ID	2011	2012	2013
A1	1	0	1
A2	2	1	3
A3	0	1	2

Conversion from wide to long formats can be handled using the pandas wide_to_long method.

(This method will be more integrated in the future)

4.4 Other Formats

As mentioned, a design choice is that data ingestion of transitionMatrix is via a pandas dataframe so other formats can be handled with additional code by the user. If there is a format that you repeatedly encounter submit an issue with your desired format / transformation suggestion.

CHAPTER 5

Datasets

The transitionMatrix distribution includes a number of datasets to support testing / training objectives. Datasets come in two main types:

- State Transition Data (used in estimation). There are both dummy (synthetic) examples and some actual data. Transition data are usually in CSV format.
- Transition Matrices and Multi-period Sets of matrices (again both dummy and actual examples). Transition matrices are usually in JSON format.

5.1 State Transition Data

The scripts are located in examples/python. For testing purposes all examples can be run using the run_examples.py script located in the root directory. Some scripts have an example flag that selects alternative input data or estimators.

Table 1: List of Transition Datasets

File	Format	Events	Entities	States	Generator	Description
rating_data_raw.csv	Compact	4000	1829	9	Extract	A typical credit rating dataset
rating_data.csv	Compact	3780	1642	9	Data cleaning script	A typical credit rating dataset
scenario_data.csv	Compact	550	50	5		
synthetic_data.csv	Compact	100	10	2		
synthetic_data1.csv	Compact	100	1	4	Generator(=1)	DURATION TYPE DATASETS (Compact format)
synthetic_data2.csv	Compact	10000	1000	2	Generator(=2)	DURATION TYPE DATASETS (Compact format)
synthetic_data3.csv	Compact	2000	100	7	Generator(=3)	DURATION TYPE DATASETS (Compact format)
synthetic_data4.csv	Compact	10000	1000	8	Generator(=4)	Cohort type dataset (Generic Rating Matrix). Offers a semi-realistic example
synthetic_data5.csv	Compact	50000	10000	3	Generator(=5)	Large cohort type dataset useful for testing convergence
synthetic_data6.csv	Compact	20000	1000	2	Generator(=6)	COHORT TYPE DATASETS
synthetic_data7.csv	Canonical	1295	1000	8	Generator(=7)	Duration type datasets in Long Format
synthetic_data8.csv	Canonical	10001	10000	2	Generator(=8)	Duration type datasets in Long Format
synthetic_data9.csv	Canonical	1338	1000	8	Generator(=9)	Duration type datasets in Long Format
synthetic_data10.csv	Canonical	12002	2000	9	Generator(=10)	Credit Rating Migrations in Long Format / Compact Form
test.csv	Compact	14	7	3		

5.2 Transition Matrices

- generic_monthly
- generic_multiperiod
- JLT
- sp 2017

CHAPTER 6

Preprocessing

The preprocessing stage includes preparatory steps leading up to the matrix *Estimation* to produce a transition matrix (or matrix set).

The precise steps required depend on the sources of data, the nature of data, use specific requirements (best practices, regulation etc) and, not least, the desired estimation method.

6.1 State Spaces

A State Space is a fundamental concept in probability theory and computer science representing the possible configurations for a modelled system

The StateSpace object stores a state space structure as a List of tuples. The first two elements of each tuple contain the index (base-0) and label of the state space respectively.

Additional fields are reserved for further characterisation

6.1.1 Example: Map credit ratings between systems

- Script: state_space_operations.py

Example workflows for converting data from one credit rating system to another using an established mapping table

6.2 Cohorts

Organizing data in cohorts can be an important step in understating transition data or towards applying a *Cohort Estimator*. Cohorts in this context are understood as the grouping of entities within a temporal interval. For example in a credit rating analysis context cohorts could be groups of annual observations. The implication of cohorting data is that the more granular information embedded in a more precise timestamp is not relevant. It is also possible that input data are only available in cohort form (when the precise timestamp information is not recorded at the source)

Note: Cohorting can bias the estimation in various subtle ways, so it is important that any procedure is well documented.

6.2.1 Cohorting Utilities

Cohorting utilities are part of *Preprocessing*. Presently the core algorithm is implemented in `transitionMatrix.utils.preprocessing.bin_timestamps()`.

6.2.2 Intermediate Cohort Data Formats

The cohort data format is a tabular representation of time series data that records the states (measurements) of multiple entities. Its defining characteristic is that each table row contains data pertaining to one entity at one point in time.

The *canonical form* used as input to duration based estimators uses normalized timestamps (from 0 to T_max, where T_max is the last timepoint) and looks as follows:

ID	Time	From	To
1	1.1	0	1
1	2.0	1	2
1	3.4	2	3
1	4.0	3	2
2	1.2	0	1
2	2.4	1	2
2	3.5	2	3

6.2.3 Cohorting Examples

Cohorting Example 1

An example with limited data (dataset contains only one entity). It is illustrated in script examples/python./matrix_from_duration_data.py with example flag set to 1. Input data set is synthetic_data1.csv

The state space is as follows (for brevity we work directly with the integer representation)

```
[('0', "A"), ('1', "B"), ('2', "C"), ('3', "D")]
```

The cohorting algorithm that assigns the last state to the cohort results in the following table. We notice that there is alot of movement inside each cohort (high count) and that only two of the states are represented at the cohort level (0 and 1).

```
   ID  Cohort State       Time  Count
0   0       0     0   2.061015   21.0
1   0       1     1   4.400105   14.0
2   0       2     0   6.665899   28.0
3   0       3     0   8.842277   14.0
4   0       4     0  11.111733   21.0
5   0       5     0  11.182184    2.0
```

CHAPTER 7

Credit Ratings

Working with credit data is a core use case of transitionMatrix. Functionality that is specific to credit ratings is generally grouped in the **credit ratings** subpackage (although the distinction of what is generic and what credit specific is not always clear).

The following sections document various credit rating related activities. General documentation about credit rating systems

7.1 Predefined Rating Scales

The transitionMatrix package supports a variety of credit rating scales. They are grouped together in `transitionMatrix.creditratings.creditsystems`.

The key ones are described here in more detail.

7.1.1 Rating Scales currently covered

The focus of the current selection is on *long-term issuer* ratings scales (others will be added):

- AM Best Europe-Rating Services Ltd.
- ARC Ratings S.A.
- Cerved Rating Agency S.p.A.
- Creditreform Rating AG
- DBRS Ratings Limited
- Fitch Ratings
- Moody's Investors Service
- Scope Ratings AG
- Standard & Poor's Ratings Services

7.1.2 Data per Scale

Each rating scale is a StateSpace (see *State Spaces*) and thus inherits the attributes and methods of that object, namely:

- The entity defining the scale (the originating entity)
- The full name of the scale (as most originators of rating scales offer multiple scales with different meaning an/or use)
- The definition of the scale (as a list of tuples in the form [('0', 'X1'), ... , ('N-1', 'XN)] where X are the symbols used to denote the credit state
- The CQS (credit quality step) mapping of the scale as defined by regulatory authorities (see next section)

7.1.3 CQS Mappings

The Credit Quality Step (CQS) denotes a standardised indicator of Credit Risk that is recognized in the European Union

- The CQS Credit Rating Scale is based on numbers, ranging from 1 to 6.
- 1 is the highest quality, 6 is the lowest quality

The European Supervisory Authorities maintain mappings between credit rating agencies and CQS

Note: Consult the original documents from definitive mappings available at the EBA Website

The Rating Agency State Spaces and mappings are obtained from the latest (20 May 2019) Regulatory Reference:

```
JC 2018 11, FINAL REPORT: REVISED DRAFT ITS ON THE MAPPING OF ECAIS' CREDIT␣
↪ASSESSMENTS UNDER CRR
```

Example of Label Conversion

Convert labels between credit rating scales

```
Some Basics
================================================================================
The States of our starting scale:  ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
The State Labels:  ['AAA', 'AA', 'A', 'BBB', 'BB', 'B', 'CCC', 'CC', 'C', 'R', 'SD/D']
The Full Description:  [('0', 'AAA'), ('1', 'AA'), ('2', 'A'), ('3', 'BBB'), ('4', 'BB'), ('5', 'B'), ('6',
  'CCC'), ('7', 'CC'), ('8', 'C'), ('9', 'R'), ('10', 'SD/D')]

Convert labels to other rating scales scales
================================================================================
AAA  ----> ( Aaa AAA )
AA  ----> ( Aa2 AA )
A  ----> ( A2 A )
BBB  ----> ( Baa2 BBB )
BB  ----> ( Ba2 BB )
B  ----> ( B2 B )
CCC  ----> ( Caa2 CCC )
CC  ----> ( Ca CC )
C  ----> ( Ca CC )

Convert data to other scales
================================================================================
Input S&P Labels:
['AA' 'BB' 'CC' 'AA' 'CC' 'BB' 'BB' 'CCC' 'SD/D' 'AA' 'CC' 'BBB' 'R' 'C'
 'C' 'SD/D' 'SD/D' 'A' 'BB' 'A' 'C' 'BB' 'AA' 'BB' 'CCC' 'R' 'CC' 'A' 'C'
 'BB' 'B' 'B' 'BBB' 'SD/D' 'A' 'AAA' 'C' 'AA' 'SD/D' 'CCC' 'R' 'CCC' 'CC'
 'B' 'BBB' 'BBB' 'BBB' 'SD/D' 'CC' 'A' 'A' 'CC' 'BBB' 'AA' 'BB' 'SD/D'
 'AA' 'C' 'CC' 'A' 'AAA' 'C' 'BBB' 'R' 'B' 'B' 'CC' 'AA' 'BBB' 'AA' 'AA'
 'C' 'C' 'AA' 'BBB' 'AAA' 'BBB' 'BBB' 'R' 'BB' 'BBB' 'CC' 'A' 'B' 'A'
 'CCC' 'AAA' 'CC' 'SD/D' 'BBB' 'A' 'BBB' 'BB' 'CC' 'AA' 'B' 'C' 'AA' 'BBB'
 'CCC']

Output CQS Labels:
['1', '4', '6', '1', '6', '4', '4', '6', '6', '1', '6', '3', '6', '6', '6', '6', '6', '2', '4', '2', '6',
  '4', '1', '4', '6', '6', '6', '2', '6', '4', '5', '5', '3', '6', '2', '1', '6', '1', '6', '6', '6', '6',
  '6', '5', '3', '3', '3', '6', '6', '2', '2', '6', '3', '1', '4', '6', '1', '6', '6', '2', '1', '6', '3',
  '6', '5', '5', '6', '1', '3', '1', '1', '6', '6', '1', '3', '1', '3', '3', '6', '4', '3', '6', '2', '5',
  '2', '6', '1', '6', '6', '3', '2', '3', '4', '6', '1', '5', '6', '1', '3', '6']
```

7.2 Withdrawn Ratings

Withdrawn ratings are a common issue that needs to be handled in the context of estimating transition matrices. See right censoring issues

7.2.1 Adjust NR (Not Rated) States

Adjusting for NR states can be done via the `transitionMatrix.model.TransitionMatrix.remove()` method.

Single Period Matrix

Example of using transitionMatrix to adjust the (not-rated) NR state. Input data are the Standard and Poor's historical data (1981 - 2016) for corporate credit rating migrations. Example of handling

- Script: examples/python/adjust_nr_states.py

7.2.2 Multi-period Matrix

- Script: examples/python/fix_multiperiod_matrix.py

Example of using transitionMatrix to detect and solve various pathologies that might be affecting transition matrix data

7.3 Credit Curves

A Credit Curve denotes a grouping of credit risk metrics (parameters) that provide estimates that a legal entity experiences a Credit Event over different (an increasing sequence of longer) time periods. See Credit Curves

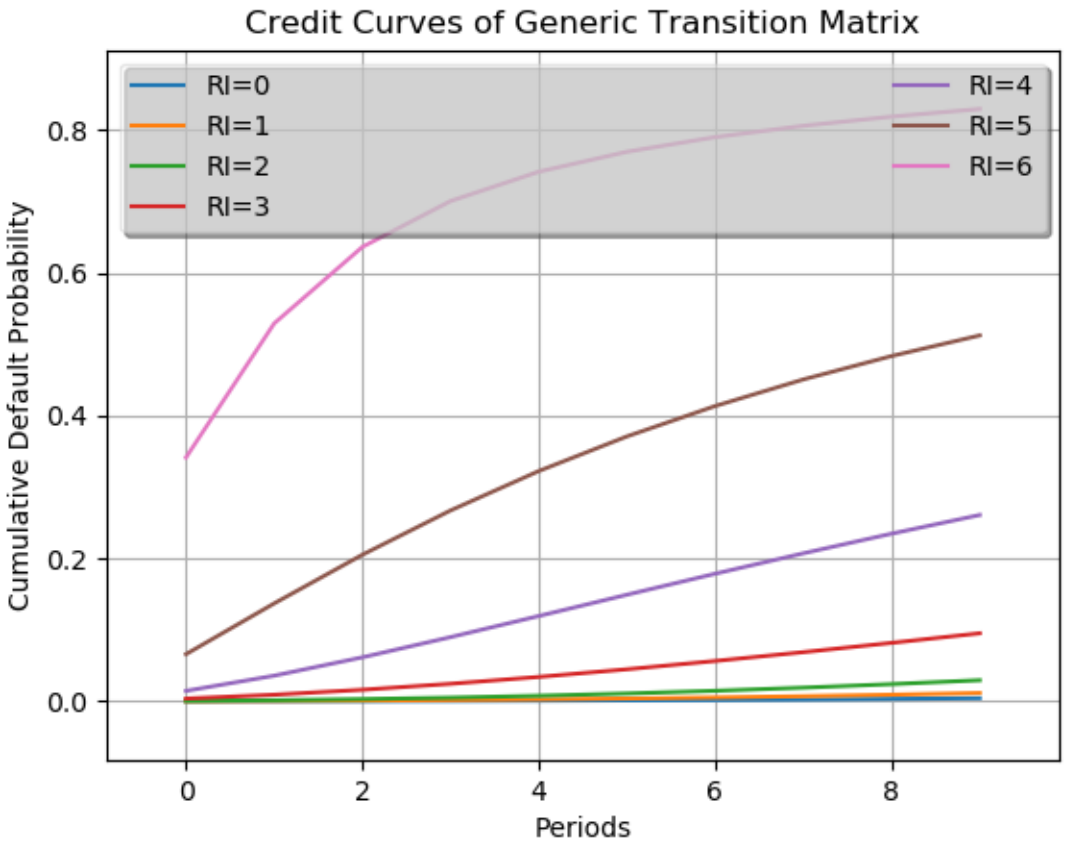
A multi-period matrix and a credit curve are closely related objects (under some circumstances the later can be thought of as a subset of the former). The transitionMatrix package offers the following main functionality concerning credit curves:

- The `transitionMatrix.creditratings.creditcurve.CreditCurve` class for storing and working with credit curves
- The `transitionMatrix.model.TransitionMatrixSet.default_curves()` transitionMatrixSet method that extracts from a matrix set the default curve

7.3.1 Example: Calculate and Plot Credit Curves

Example of using transitionMatrix to calculate and visualize multi-period

- Script: examples/python/credit_curves.py



CHAPTER 8

Estimation

The estimation of a transition matrix is one of the core functionalities of transitionMatrix. Several methods and variations are available in the literature depending on aspects such as:

- The nature of the observations / data (e.g., whether temporal homogeneity is a valid assumption)
- Whether or not there are competing risk effects
- Whether or not observations have coincident values
- Treating the Right-Censorship of observations (Outcomes beyond the observation window)
- Treating the Left-Truncation of observations (Outcomes prior to the the observation window)

8.1 Estimator Types

- **Cohort Based Methods** that group observations in cohorts
- **Duration** (also Hazard Rate or Intensity) Based Methods that utilize the actual duration of each state

The main estimators currently implemented are as follows:

8.1.1 Simple Estimator

The estimation of a transition matrix is one of the core functionalities of transitionMatrix. The two main estimators currently implemented are:

8.1.2 Cohort Estimator

A cohort estimator (more accurately discrete time estimator) is class of estimators of multi-state transitions that is a simpler alternative to Duration type estimators

Estimate a Transition Matrix from Cohort Data

Example workflows using transitionMatrix to estimate a transition matrix from data that are already grouped in cohorts

- Script: examples/python/matrix_from_cohort_data.py
- Example ID: 3

```
data = pd.read_csv(dataset_path + 'synthetic_data6.csv', dtype={'State': str})
sorted_data = data.sort_values(['ID', 'Timestep'], ascending=[True, True])
myState = tm.StateSpace()
myState.generic(2)
print(myState.validate_dataset(dataset=sorted_data))
myEstimator = es.CohortEstimator(states=myState, ci={'method': 'goodman', 'alpha': 0.
↪05})
result = myEstimator.fit(sorted_data)
myMatrixSet = tm.TransitionMatrixSet(values=result, temporal_type='Incremental')

myEstimator.print(select='Counts', period=0)
myEstimator.print(select='Frequencies', period=18)
```

8.1.3 Aalen-Johansen Estimator

The Aalen-Johansen estimator is a multi-state (matrix) version of the Kaplan–Meier estimator for the hazard of a survival process. The estimator can be used to estimate the transition probability matrix of a Markov process with a finite number of states. See

Whichever the estimator choice, the outcome of the estimation is an *Empirical Transition Matrix* (or potentially a matrix set)

8.1.4 Implementation Notes

- All estimators derive from the highest level *BaseEstimator* class.
- Duration type estimators derive from the *DurationEstimator* class

8.2 Estimation Examples

The first example of estimating a transition matrix is covered in the *Getting Started* section. Here we have a few more examples:

8.2.1 Estimation Example 1

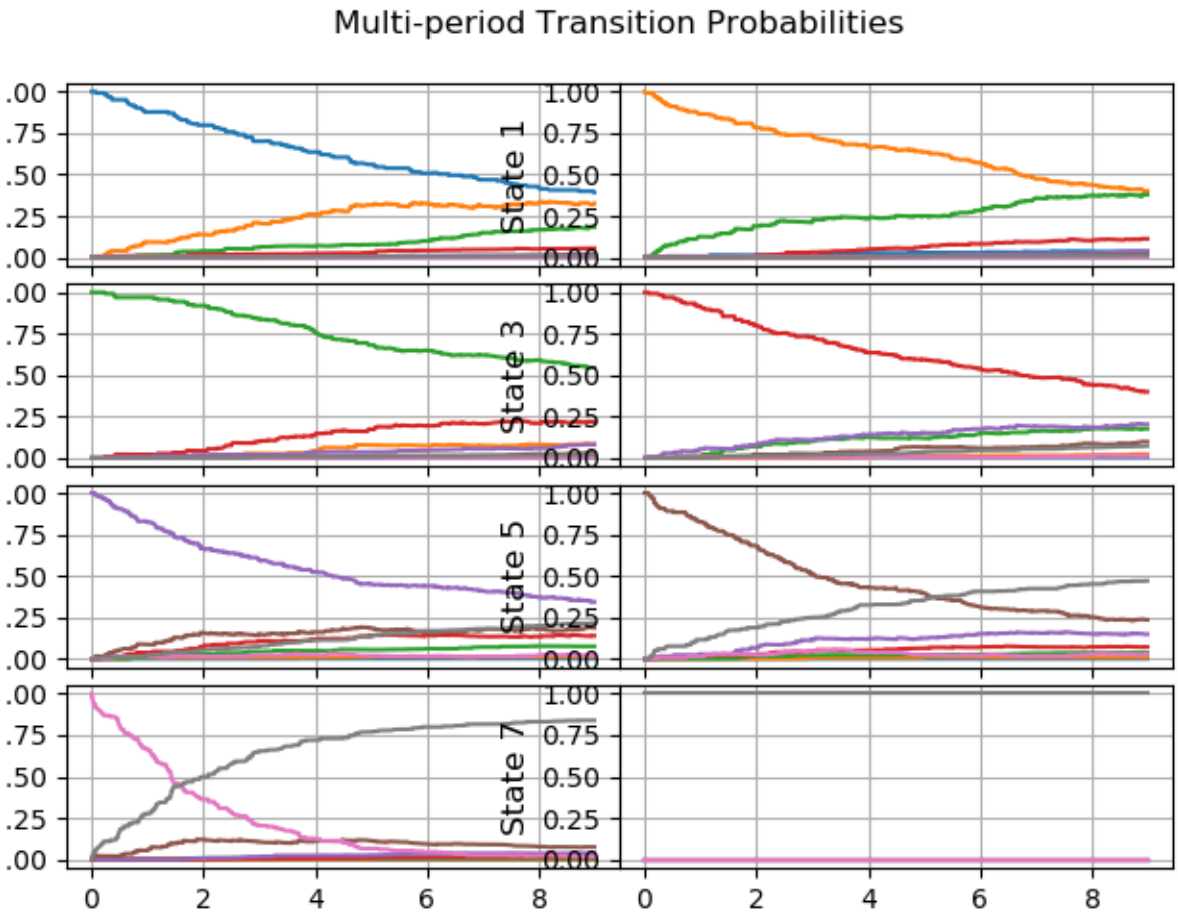
Example workflows using transitionMatrix to estimate an empirical transition matrix from duration type data. The datasets are produced using examples/generate_synthetic_data.py This example uses the Aalen-Johansen estimator

- Script: examples/python/empirical_transition_matrix.py

By setting the example variable the script covers a number of variations:

- Version 1: Credit Rating Migration example
- Version 2: Simple 2x2 Matrix for testing
- Version 3: Credit Rating Migration example with timestamps in raw date format

Plot of estimated transition probabilities



8.2.2 Estimation Example 2

Example workflows using transitionMatrix to estimate a transition matrix from data that are in duration format. The datasets are first grouped in period cohorts

- Script: examples/python/matrix_from_duration_data.py

CHAPTER 9

Post-processing

The post-processing stage includes steps and activities after the estimation of a transition matrix. The precise steps required depend on specific circumstances but might involve some of the following:

- "Fixing" a matrix by correcting deficiencies linked to data quality
- Obtaining the infinitesimal generator of a matrix, a powerful tool for further analysis
- Working with multi-period matrices
- Visualizing transition datasets and transition frequencies

9.1 Basic Operations

The core TransitionMatrix object implements a typical (one period) transition matrix. It supports a variety of operations (more details are documented in the API section)

- Initialize a matrix (from data, predefined matrices etc)
- Validate a matrix
- Attempt to fix a matrix
- Compute generators, powers etc.
- Print a matrix
- Output to json/csv/xlsx formats
- Output to html format

9.1.1 Simple Operation Examples

Note: The script examples/python/matrix_operations.py contains the below and plenty more simple single matrix examples

Initialize a matrix with values

There is a growing list of ways to initialize a transition matrix

- Initialize a generic matrix of dimension n
- Any list can be used for initialization (but not all shapes are valid transition matrices!)
- Any numpy array can be used for initialization (but not all are valid transition matrices!)
- Values can be loaded from json or csv files
- The transitionMatrix.creditratings.predefined module includes a number of predefined matrices

```
A = tm.TransitionMatrix(values=[[0.6, 0.2, 0.2], [0.2, 0.6, 0.2], [0.2, 0.2, 0.6]])
print(A)
A.print_matrix(format_type='Standard', accuracy=2)

[[0.6 0.2 0.2]
 [0.2 0.6 0.2]
 [0.2 0.2 0.6]]

0.60 0.20 0.20
0.20 0.60 0.20
0.20 0.20 0.60

A.print_matrix(format_type='Standard', accuracy=2)

60.0% 20.0% 20.0%
20.0% 60.0% 20.0%
20.0% 20.0% 60.0%
```

Both the intrinsic print function and the specific print_matrix will print you the matrix, but the print_matrix method clearly aims to present the values in a more legible formats.

General Matrix Algebra

Note: All standard numerical matrix operations are available as per the numpy API.

Some example operations that leverage the underlying numpy API:

```
E = tm.TransitionMatrix(values=[[0.75, 0.25], [0.0, 1.0]])
print(E.validate())
# ATTRIBUTES
# Getting matrix info (dimensions, shape)
print(E.ndim)
print(E.shape)
# Obtain the matrix transpose
print(E.T)
# Obtain the matrix inverse
```

(continues on next page)

(continued from previous page)

```
print(E.I)
# Summation methods:
# - along columns
print(E.sum(0))
# - along rows
print(E.sum(1))
# Multiplying all elements of a matrix by a scalar
print(0.01 * A)
# Transition Matrix algebra is very intuitive
print(A * A)
print(A ** 2)
print(A ** 10)
```

9.1.2 Validating, Fixing and Characterizing a matrix

Validate a Matrix

The validate() method of the object checks for required properties of a valid transition matrix:

1. check squareness
2. check that all values are probabilities (between 0 and 1)
3. check that all rows sum to one

```
C = tm.TransitionMatrix(values=[1.0, 3.0])
print(C.validate())

[('Matrix Dimensions Differ: ', (1, 2))]
```

Characterise a Matrix

The characterise() method attempts to characterise a matrix

1. diagonal dominance

9.2 Working with an actual matrix

The core capability of transitionMatrix is to produce estimated matrices but getting a realistic example requires quite some work. In this section we assume we have estimated one.

Lets look at a realistic example from the JLT paper

```
# Reproduce JLT Generator
# We load it using different sources
E = tm.TransitionMatrix(values=JLT)
E_2 = tm.TransitionMatrix(json_file=dataset_path + "JLT.json")
E_3 = tm.TransitionMatrix(csv_file=dataset_path + "JLT.csv")
# Lets check there are no errors
Error = E - E_3
print(np.linalg.norm(Error))
# Lets look at validation and generators"
```

(continues on next page)

(continued from previous page)

```
# Empirical matrices will not satisfy constraints exactly
print(E.validate(accuracy=1e-3))
print(E.characterize())
print(E.generator())
Error = E - expm(E.generator())
# Frobenious norm
print(np.linalg.norm(Error))
# L1 norm
print(np.linalg.norm(Error, 1))
# Use pandas style API for saving to files
E.to_csv("JLT.csv")
E.to_json("JLT.json")
```

9.3 Multi-Period Transitions

Th transitionMatrix package adopts a *multi-period paradigm* that is more general than a Markov-Chain framework that imposes the Markov assumption over successive periods. In this direction, the **TransitionMatrixSet object** stores a family of TransitionMatrix objects as a time ordered list. Besides basic storage this structure allows a variety of simultaneous operations on the collection of related matrices

There are two basic representations of the a multi-period set of transitions:

- The first (*cumulative form*) is the most fundamental. Each successive (k-th) element stores transition rates from an initial time to timepoint k. This could be for example the input of an empirical transition matrix dataset
- In the second (*incremental form*) successive elements store transition rates from timepoint k-1 to timepoint k.

The TransitionMatrixSet class allows converting between the two representations

9.3.1 Matrix *Set* Operations

- Script: matrix_set_operations.py

Contains examples using transitionMatrix to perform various transition matrix **set** operations (Multi-period measurement context)

Default Curves

Absorbing states (in credit risk context a borrower default) are particularly important therefore some specific functionality to isolate the corresponding default rate *curve*. (See Also the CreditCurve object)

9.4 Visualization

transitionMatrix aims to support native (Python based) visualization of various transition related datasets using matplotlib and other native python visualization libraries.

Note: The visualization functionality is not yet refactored into a reusable API. For now the visualization functionality is implemented separately as a demo script.

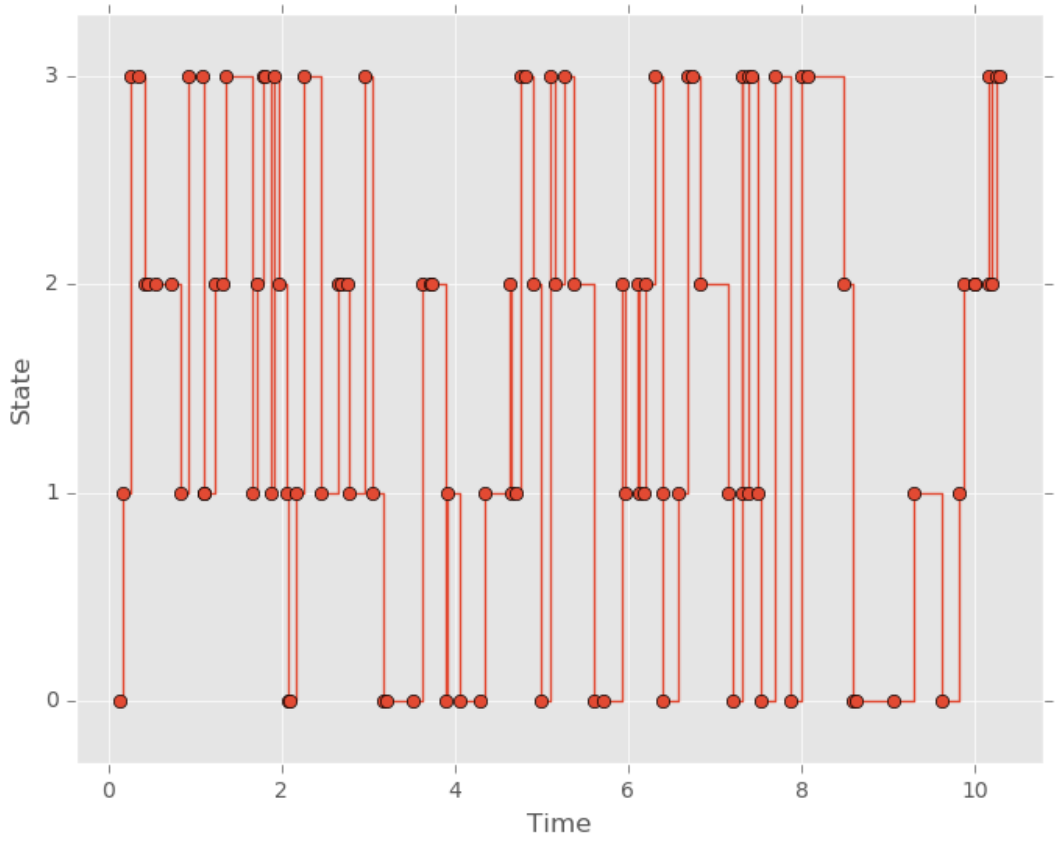
9.4.1 Visualization Examples

Example workflows using transitionMatrix to generate visualizations of migration phenomena

- Script: examples/python/generate_visuals.py

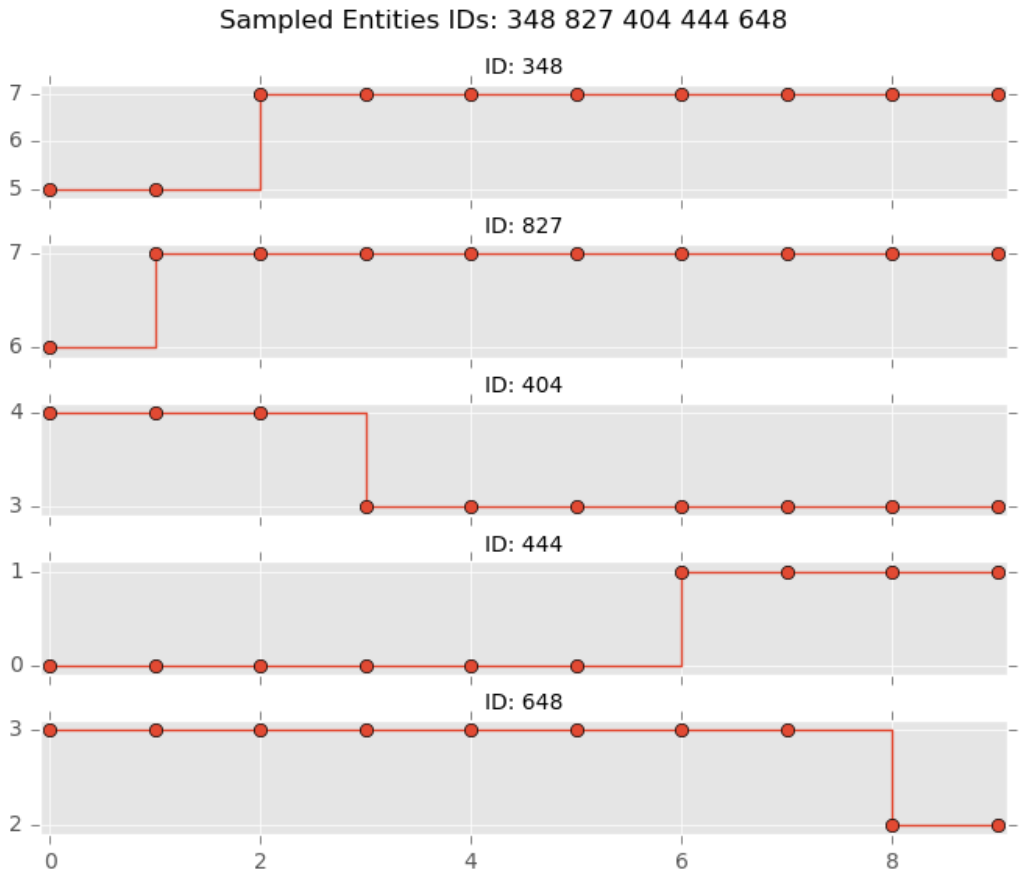
Example 1

Plotting the state space trajectory of a single entity



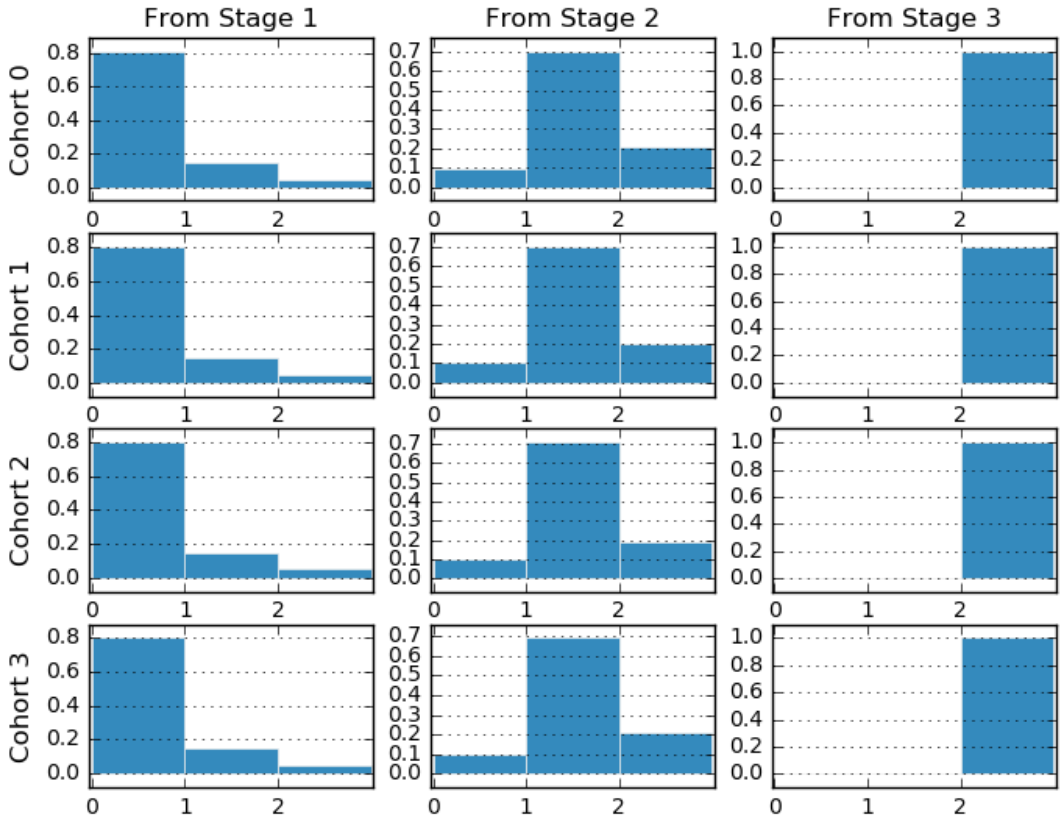
Example 2

Plotting the state space trajectory of multiple entities



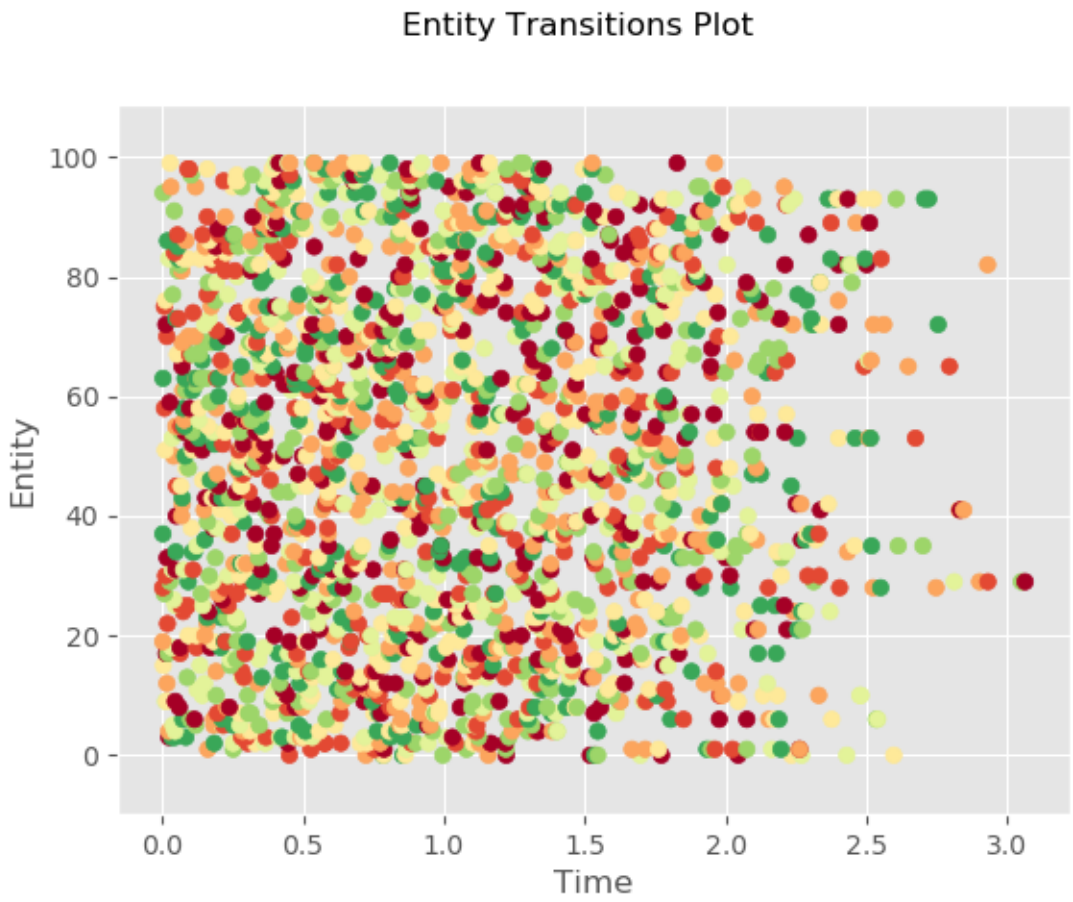
Example 3

Histogram plot of transition frequencies



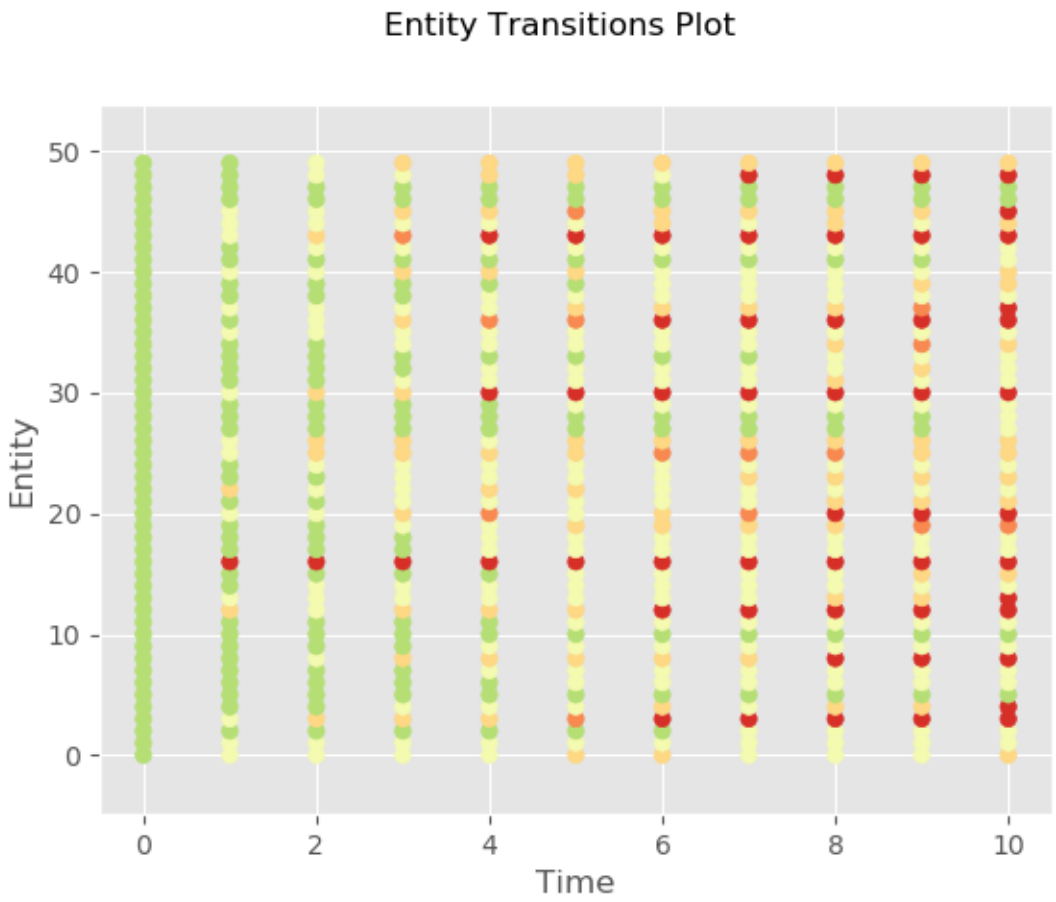
Example 4

Colored scatterplot of entity transitions over time



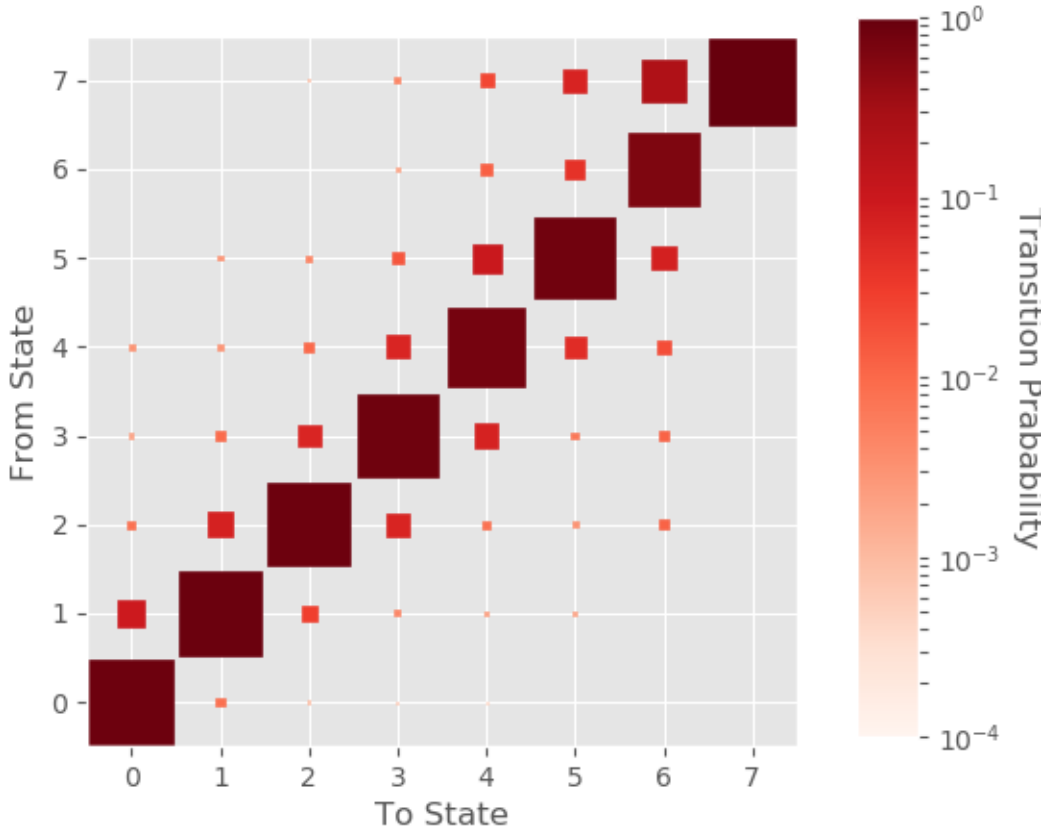
Example 5

Colored scatterplot of entity transitions over time (alternative form)



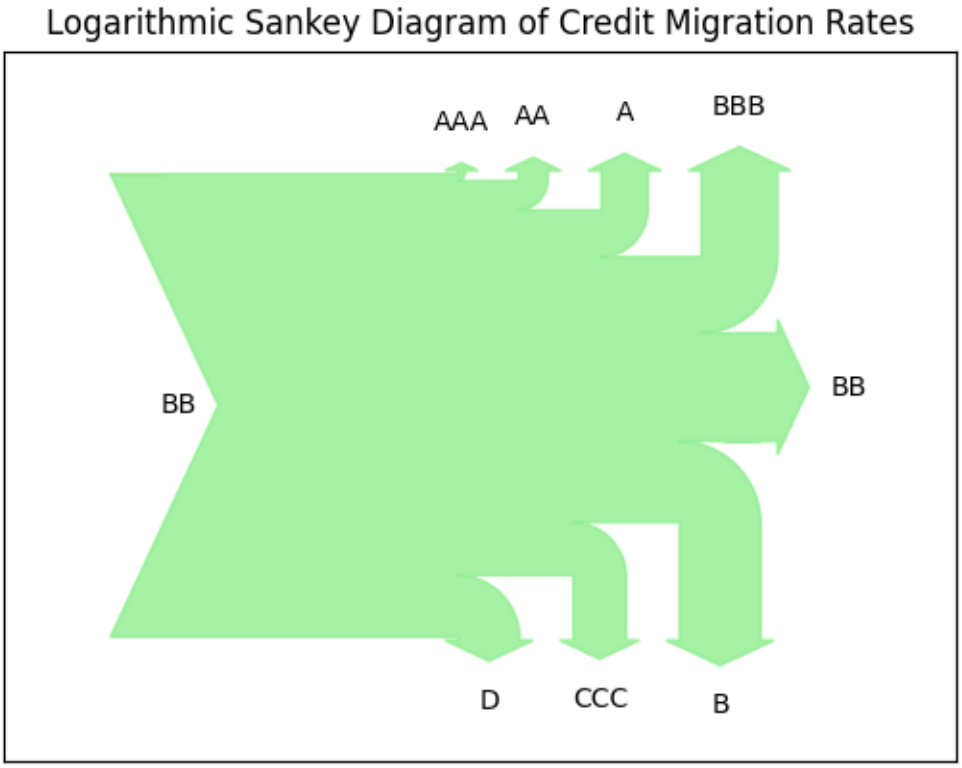
Example 6

Visualize a transition matrix using Hinton-style visual



Example 7

Visualize a transition matrix using a sankey visual (a logarithmic adaptation that is useful for qualitative insight)


Logarithmic Sankey Diagram of Credit Migration Rates
AAA
AA
A
BBB
BB
BB
D
CCC
B

CHAPTER 10

Data Generators

The transitionMatrix distribution includes a number of data generators to support testing / training objectives.

- **exponential_transitions**: Generate continuous time events from exponential distribution and uniform sampling from state space. Suitable for testing cohorting algorithms and duration based estimators.
- **markov_chain**: Generate discrete events from a markov chain matrix in Compact data format. Suitable for testing cohort based estimators
- **long_format**: Generate continuous events from a markov chain matrix in Long data format. Suitable for testing duration based estimators
- **portfolio_lables**: Generate a collection of credit rating states emulating a snapshot of portfolio data. Suitable for mappings and transformations of credit rating states

Note: Do not confuse *data generators* with *matrix generators*

10.1 Data Generation Examples

All data data generation examples are in script examples/python/generate_synthetic_data.py

CHAPTER 11

Federation

11.1 Credit Rating Ontology

The Credit Ratings Ontology is a framework that aims to represent and categorize knowledge about Credit Rating Agencies and related data (Credit Ratings) using semantic web information technologies.

This is a new project, related resources can be found here:

- Online documentation
- Blog post
- Course
- Repo with ontology usage examples

Note: transitionMatrix functionality to federate semantically annotated credit data is planned

CHAPTER 12

Usage Examples

The examples directory includes both **standalone python scripts** and **jupyter notebooks** to help you get started. (NB: Currently there are more scripts than notebooks).

A selection of topics covered:

- Generating transition matrices from data (using various estimators)
- Manipulating transition matrices
- Computing and visualizing credit curves corresponding to a set of transition matrices
- Mapping rating states between different rating systems

12.1 Python Scripts

The scripts are located in examples/python. For testing purposes all examples can be run using the run_examples.py script located in the root directory. Some scripts have an example flag that selects alternative input data or estimators.

Table 1: List of Example Scripts

Script Name	Flag	Input Data	Description
adjust_nr_state.py	1		Adjust the NR (not-rated) statistics.
adjust_nr_state.py	2		Adjust the NR (not-rated) statistics.
credit_curves.py			Compute and Visualize credit curves
characterize_datasets.py			Load the available datasets and compute various statistics
compare_estimators.py		synthetic_data4.csv	Compare the cohort and aalen-johansen estimators on a discrete timestep sample
data_cleaning_example.py		rating_data_raw.csv	Prepare transition data sets (data cleansing) using some provided methods
deterministic_paths.py			Create a transition dataset by replicating give trajectories through a graph
empirical_transition_matrix.py		synthetic_data7.csv	Credit Rating Migration example

Continued on next page

Table 1 – continued from previous page

Script Name	Flag	Input Data	Description
empirical_transition_matrix.py	2	synthetic_data8.csv	Simple 2x2 Matrix for testing
empirical_transition_matrix.py	3	synthetic_data9.csv	Credit Rating Migration example with timestamps in raw date format
estimate_matrix.py		rating_data.csv	An end-to-end example of estimating a credit rating matrix from historical data
fix_multiperiod_matrix.py		sp_1981-2016.csv	Detect and solve various pathologies that might be affecting transition matrix data
generate_full_multiperiod_set.py		NR_adjusted.json	Use infinitesimal generator methods to generate a full multi-period matrix set.
generate_synthetic_data.py	1		Generate synthetic data. The first set of examples produces duration type data.
generate_synthetic_data.py	2		The second set of examples produces cohort type data using markov chain simulation
generate_synthetic_data.py	3		The second set of examples produces cohort type data using markov chain simulation
generate_visuals.py	6	JLT.json	Plot Transition Probabilities
generate_visuals.py	7	JLT.json	Logarithmic Sankey Diagram of Credit Migration Rates
generate_visuals.py	5	scenario_data.csv	Plot Entity Transitions Plot
generate_visuals.py	1	synthetic_data1.csv	Step Plot of a single observation
generate_visuals.py	4	synthetic_data3.csv	Entity Transitions Plot
generate_visuals.py	2	synthetic_data4.csv	Step Plot of individual observations
generate_visuals.py	3	synthetic_data5.csv	Histogram Plots of transition frequencies
matrix_from_cohort_data.py	3	synthetic_data4.csv	S&P Style Credit Rating Migration Matrix
matrix_from_cohort_data.py	2	synthetic_data5.csv	IFRS 9 Style Migration Matrix (Large sample for testing)
matrix_from_cohort_data.py	1	synthetic_data6.csv	Simplest Absorbing Case for validation
matrix_from_duration_data.py	1	synthetic_data1.csv	Duration example with limited data (dataset contains only one entity)
matrix_from_duration_data.py	2	synthetic_data2.csv	Duration example n entities with ~10 observations each, [0,1] state, 50%/50% transition matrix
matrix_from_duration_data.py	3	synthetic_data3.csv	
matrix_lendingclub.py			Estimate a matrix from LendingClub data. Input data are in a special cohort format as the published datasets have some limitations
matrix_operations.py			Perform various transition matrix operations illustrating the matrix algebra
matrix_set_lendingclub.py			Estimate a matrix from LendingClub data. Input data are in a special cohort format as the published datasets have some limitations
matrix_set_operations.py			Perform operations with multi-period transition matrix sequences
state_space_operations.py			Examples working with state spaces (mappings)

12.2 Jupyter Notebooks

- Adjust_NotRated_State.ipynb
- Matrix_Operations.ipynb
- Monthly_from_Annual.ipynb

12.3 Open Risk Academy Scripts

Additional examples are available in the Open Risk Academy course Analysis of Credit Migration using Python TransitionMatrix. The scripts developed in the course are available here

CHAPTER 13

API

The transitionMatrix package structure and API.

Warning: The library is still being expanded / refactored. Significant structure and API changes are likely.

13.1 transitionMatrix Package

The core module

13.1.1 transitionMatrix Classes

TransitionMatrix

TransitionMatrixSet

EmpiricalTransitionMatrix

Todo: This is future functionality

13.2 transitionMatrix Subpackages

13.2.1 Estimators SubPackage

This subpackage implements the various estimators

transitionMatrix.estimators.simple_estimator module

transitionMatrix.estimators.cohort_estimator module

transitionMatrix.estimators.aalen_johansen_estimator module

transitionMatrix.estimators.kaplan_meier_estimator module

Todo: This is future functionality

13.2.2 State Spaces SubPackage

This subpackage implements state space functionality

transitionMatrix.statespaces.statespace module

13.2.3 Credit Ratings SubPackage

This subpackage collects credit rating specific functionality

transitionMatrix.creditratings.creditcurve module

transitionMatrix.creditratings.creditsystems module

transitionMatrix.creditratings.predefined module

13.2.4 Generators SubPackage

This subpackage implements test data generation

transitionMatrix.generators contents

13.2.5 Visualization subpackage

This subpackage implements visualization functionality

Warning: not yet implemented

transitionMatrix.visualization contents

13.2.6 Utilities SubPackage

This subpackage collects various utilities

transitionMatrix.utils.converters module

transitionMatrix.utils.preprocessing module

CHAPTER 14

Testing

Testing transitionMatrix has two major components:

- normal code testing aiming to certify the correctness of code execution
- algorithm testing aiming to validate the correctness of algorithmic implementation

Note: In general algorithmic testing is not as precise as code testing and may be more subject to uncertainties such as numerical accuracy. To make those tests as revealing as possible transitionMatrix implements a number of standardized *round-trip tests*:

- starting with a matrix
- generating compatible data
- estimate a matrix from the data
- comparing the values of input and estimated matrices

14.1 Running all the examples

Running all the examples is a quick way to check that everything is installed properly, all paths are defined etc. At the root of the distribution:

```
python3 run_examples.py
```

The file simply iterates and executes a standalone list of *Usage Examples*.

```
filelist = ['adjust_nr_state', 'credit_curves', 'empirical_transition_matrix', 'fix_
↪multiperiod_matrix', 'generate_synthetic_data', 'generate_visuals', 'matrix_from_
↪cohort_data', 'matrix_from_duration_data', 'matrix_lendingclub', 'matrix_set_
↪lendingclub', 'matrix_operations', 'matrix_set_operations']
```

Warning: The script might generate a number of files / images at random places within the distribution

14.2 Test Suite

The testing framework is based on unittest. Before you get started and depending on how you obtained / installed the library check:

- If required adjust the source directory path in transitionMatrix/__init__
- Unzip the data files in the datasets directory

Then run all tests

```
python3 test.py
```

For an individual test:

```
pytest tests/test_TESTNAME.py
```

CHAPTER 15

Roadmap

transitionMatrix is an ongoing project. Several significant extensions are already in the pipeline. transitionMatrix aims to become the most intuitive and versatile tool to analyse discrete transition data. The **Roadmap** lays out upcoming steps / milestones in this journey. The **Todo** list is a more granular collection of outstanding items.

You are welcome to contribute to the development of transitionMatrix by creating Issues or Pull Requests on the github repository. Feature requests, bug reports and any other issues are welcome to log at the Github Repository

Discussing general usage of the library is happening here

15.1 0.5

The 0.5 will be the next major release (still considered alpha) that will be available e.g. on PyPI

15.2 0.4.X

The 0.4.X family of updates will focus on rounding out and (above all) documenting a number of functionalities already introduced

CHAPTER 16

Todo List

A list of todo items, no triaging / prioritisation implied

16.1 Core Architecture and API

- Introduce exceptions / error handling throughout
- Solve numpy.matrix deprecation (implement equivalent API in terms of ndarray)
- Complete testing framework

16.2 Input Data Preprocessing

- Handing of markov chain transition formats (single entity)
- Native handling of Wide Data Formats (concrete data sets missing)
- Generalize cohorting algorithm to user specified function

16.3 Reference Data

- Additional credit rating scales (e.g short term ratings)
- Integration with credit rating ontology

16.4 Transition Matrix Analysis Functionality

- Further validation and characterisation of transition matrices (mobility indexes)

- Generate random matrix subject to constraints
- Fixing common problems encountered by empirically estimated transition matrices

16.5 Statistical Analysis Functionality

- **Aalen Johansen Estimator**
  - Covariance calculation
  - Various other improvements / tests
- **Cohort Estimator**
  - Read Data by labels
  - Edge cases
- **Kaplan Meier Estimator NEW**
  - (link to survival frameworks)
- Duration based methods
- Bootstrap based confidence intervals

16.6 State Space package

- Multiple absorbing states (competing risks)
- Automated coarsening of states (merging of similar)

16.7 Credit Rating Related

- Import data defined according to CRO ontology
- Absorbing State Identification, Competing Risks
- Compute hazard rates
- Characterize hazard rates

16.8 Utilities

- Continuous time data generation from arbitrary chain

16.9 Further Refactoring of packages

- Introduce visualization objects / API

16.10 Performance / Big data

- Handling very large data sets, moving away from in-memory processing

16.11 Documentation

- Sphinx documentation (complete)
- Expand the jupyter notebook collection to (at least) match the standalone scripts

16.12 Releases / Distribution

- Adopt regular github/PyPI release schedule
- Conda distribution

CHAPTER 17

ChangeLog

PLEASE NOTE THAT THE API OF TRANSITION MATRIX IS STILL UNSTABLE AS MORE USE CASES / FEATURES ARE ADDED REGULARLY

17.1 v0.5.0 (21-02-2022)

- **Installation:**
  - Bump python dependency to 3.7
  - PyPI release update

17.2 v0.4.9 (04-05-2021)

- **Refactoring: All non-core functionality moved to separate directories/sub-packages**
  - credit curve stuff moved to creditratings modules
  - data generators moved to generators modules
  - etc.
- **Documentation: Major expansion (Still incomplete)**
  - Expanded Data Formats
  - Rating Scales, CQS etc
  - Listing all datasets and examples
- **Testing / Training: An interesting use case raised as issue #20**
  - Added an end-to-end example of estimating a credit rating matrix from raw data
  - Includes various data preprocessing examples

- **Datasets:**
  - rating_data.csv (cleaned up credit data)
  - synthetic_data10.csv Credit Rating Migrations in Long Format / Compact Form (for testing)
  - deterministic generator (replicate given trajectories)
- **Tests:**
  - test_roundtrip.py testing via roundtriping methods

17.3 v0.4.8 (07-02-2021)

- Documentation: Pulled all rst files in docs
- Refactoring: credit rating data moved into separate module

17.4 v0.4.7 (29-09-2020)

- Documentation: Expanded and updated description of classes
- Documentation: Including Open Risk Academy code examples
- Feature: logarithmic sankey visualization

17.5 v0.4.6 (22-05-2019)

- Feature: Update of CQS Mappings, addition of new rating scales
- Documentation: Documentation of rating scale structure and mappings
- Training: Example of mapping portfolio data to CQS

17.6 v0.4.5 (21-04-2019)

- Training: Monthly_from_Annual.ipynb (a Jupyter notebook illustrating how to obtain interpolate transition rates on monthly intervals)
- Datasets: generic_monthly.json
- Feature: print_matrix function for generic matrix pretty printing
- Feature: matrix_exponent function for obtaining arbitrary integral matrices from a given generator

17.7 v0.4.4 (03-04-2019)

- Documentation: Cleanup of docs following separation of threshold / portfolio models
- Datasets: generic_multiperiod.json
- Feature: CreditCurve class for holding credit curves

17.8 v0.4.3 (29-03-2019)

- Refactoring: Significant rearrangement of code (the threshold models package moved to portfolioAnalytics for more consistent structure of the code base / functionality)

17.9 v0.4.2 (29-01-2019)

- Feature: converter function in transitionMatrix.utils.converters to convert long form dataframes into canonical float form
- Datasets: synthetic_data9.csv (datetime in string format)
- Training: new data generator in examples/generate_synthetic_data.py to generate long format with string dates
- Training: Additional example (=3) in examples/empirical_transition_matrix.py to process long format with string dates
- Documentation: More detailed explanation of Long Data Formats with links to Open Risk Manual
- Documentation: Enabled sphinx.ext.autosectionlabel for easy internal links / removed duplicate labels

17.10 v0.4.1 (31-10-2018)

- Feature: Added functionality for conditioning multi-period transition matrices
- Training: Example calculation and visualization of conditional matrices
- Datasets: State space description and CGS mappings for top-6 credit rating agencies

17.11 v0.4.0 (23-10-2018)

- Installation: First PyPI and wheel installation options
- Feature: Added Aalen-Johansen Duration Estimator
- Documentation: Major overhaul of documentation, now targeting ReadTheDocs distribution
- Training: Streamlining of all examples
- Datasets: Synthetic Datasets in long format

17.12 v0.3.1 (21-09-2018)

- Feature: Expanded functionality to compute and visualize credit curves

17.13 v0.3 (27-08-2018)

- Feature: Addition of portfolio models (formerly portfolio_analytics_library) for data generation and testing
- Training: Added examples in jupyter notebook format

17.14 v0.2 (05-06-2018)

- Feature: Addition of threshold generation algorithms

17.15 v0.1.3 (04-05-2018)

- Documentation: Sphinx based documentation
- Training: Additional visualization examples

17.16 v0.1.2 (05-12-2017)

- Refactoring: Dataset paths
- Bugfix: Correcting requirement dependencies (missing matplotlib)
- Documentation: More detailed instructions

17.17 v0.1.1 (03-12-2017)

- Feature: TransitionMatrix model: new methods to merge States, fix problematic probability matrices, I/O API's
- Feature: TransitionMatrixSet mode: json and csv readers, methods for set-wise manipulations
- Datasets: Additional multiperiod datasets (Standard and Poors historical corporate rating transition rates)
- Feature: Enhanced matrix comparison functionality
- **Training: Three additional example workflows**
  - fixing multiperiod matrices (completing State Space)
  - adjusting matrices for withdrawn entries
  - generating full multi-period sets from limited observations

17.18 v0.1.0 (11-11-2017)

- First public release of the package

CHAPTER 18

Indexes and tables

- genindex
- modindex
- search