Machine Learning with Python/Scikit-Learn

- Application to the Estimation of Occupancy and Human Activities -

Tutorial proposed by:
manar.amayri@g-scop.grenoble-inp.fr
stephane.ploix@g-scop.grenoble-inp.fr

SIMUREX 2015: October 26th - 30th, 2015

Contents

Prerequisite

Before starting this tutorial, you need to install Python 3 and Scientific Python with numpy, scipy, matplotlib, scikit-learn and possibly iPython. The easiest way to do so is to use Pyzo http://www.pyzo.org/downloads.html that installs all at once. There are versions for Microsoft Windows (zip versions do not require administrator proviledge), Mac OS X and even Linux.

If Python 3 is already installed with Scientific Python, install the scikit-learn with the pip tool for instance: type in the command line interface pip3 install scikit−learn −U (or pip install scikit−learn −U).

Chapter 1

Objective of the tutorial

1.1 Building: a complex human-physical system

Because of constraining building standards, buildings and their appliances are becoming more and more efficient. As a result, consumption related to human activity is relatively much bigger than before. In addition, demand response in both electric and heat grids lead to variable tariffs that the occupants of living areas have to take into account in their everyday life: their actions and activities now do matter and it is becoming much more complex that before. Existing building automations compute relevant set-points for HVAC systems according to occupant calendars but it does not help for most decision occupants have to take. Indeed, the following questions may arise. When and how long opening windows, shutters, flaps or internal doors, taking account weather and energy cost? Where and when to go to a specific room? When to switch the HVAC system? What should be the set points? When to switch complementary cooling or heating appliances? When to use an electric appliance taking into account variable energy tariffs? In modern buildings, when to store/restore energy? Nowadays, complexity in buildings, where people spend most of their life, appeals decision-aided systems interacting with building managers and occupants to support their actions sometimes with automations and sometimes with suggestions of actions. These systems are called energy managers. Because of thermal inertia and of daily life cycle, energy managers have to be able to anticipate the upcoming day, at least; not being just reactive at current situation.

Among the system to be managed, living area systems are quite unique because they are highly human-machine cooperative systems. Indeed, the physical and control part provides services to occupants but occupants are also part of the system and influence it a lot with their own behavior. Human activities cannot be neglected and are part of the system state. It is composed of:

- context related to time, weather conditions, energy costs, heat gains per zone but also occupant current positions and activities
- controls related to doors, windows, flaps and shutters positions, configurations of the HVAC system and other electric appliances
- reactions related to indoor temperature and air quality, and to satisfaction of occupants regarding services provided by electric appliances.

Future building energy management systems may cover a large set of applications. It can support retrospective analyses of past periods by estimating and correlating actions and variables such as occupancy, usage of windows or heat flows through windows for instance. Using simulation means, it can extrapolate future states depending on hypothetical controls or replay past situations changing the controls. To develop all these applications, energy management systems have to embed knowledge and data models of the living area system; they have to be equipped with learning, estimation, simulation and optimization capabilities.

1.2 Estimating occupancy and human activities

Because living zone are both related to physics and to occupants, the state characterizing a zone at a given time is also related to occupants. What might matter is:

- location of occupants
- activities of occupants
- actions performed on the envelope
- actions performed on the HVAC system
- actions performed on other appliances

Estimating that human part of the state can be helpful with different regards.

Usage analysis Measuring and estimating what cannot be measured can help occupants to discover costly routines. It can be managed in replaying a past period, displaying both energy impacts and human past behaviors.

Occupant behavior modeling Estimating non-measurable variables related to human behavior can ease the tuning of reactive human activity models. These models can then be co-simulated with models of the physics to better represent human-physical zone systems.

Performances assessment Building performance is highly dependent on usage and therefore, it is difficult to guarantee performances except if human activity is monitored and meets a committed limit. Human activity estimation can help to check whether the limit is met or not. If human activities are known, intrinsic building performances could be computed such as envelope performance.

Parameter estimation Learning the behavior of a zone, or of a building, is an important issue for energy management systems that rely on models. But to properly estimate the internal power gains, the number of occupants in each zone has to be known.

Interactive management Energy management can be fully automatized but, in many situations, it is not possible (windows for instance cannot be moved automatically) or not acceptable. Then come the interactive energy management systems, which are cooperative human-machine systems. The nature of the possible interactions between a building energy management system and the occupants depends on the occupant activities. Sometimes, it is possible to interrupt activities to notify a critical situation, sometimes it is better to report in a mirroring system that the occupants may consult when they want. In this context, occupant activities matter.

1.3 What is machine learning?

This tutorial aims at illustrating the capabilities of machine learning for estimating occupancy and human activities, using Scientific Python.

According to wikipedia, machine learning is a subfield of computer science that evolved from the study of pattern recognition and computational learning theory in artificial intelligence. Machine learning explores the study and construction of algorithms that can learn from and make predictions on data. Such algorithms operate by building a model from example inputs in order to make data-driven predictions or decisions, rather than following strictly static program instructions. Machine learning is closely related to and often overlaps with computational statistics; a discipline that also specializes in prediction-making. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field.

Machine learning algorithms can figure out how to perform important tasks by generalizing from examples. This is often feasible and cost effective where manual programming is not. As more data becomes available, more ambitious problems can be tackled. As a result, machine learning is widely used in computer science and other fields.

Machine learning usually refers to the changes in systems that perform tasks associated with artificial intelligence (AI). Such tasks involve recognition, diagnosis, planning, robot control, prediction, etc...

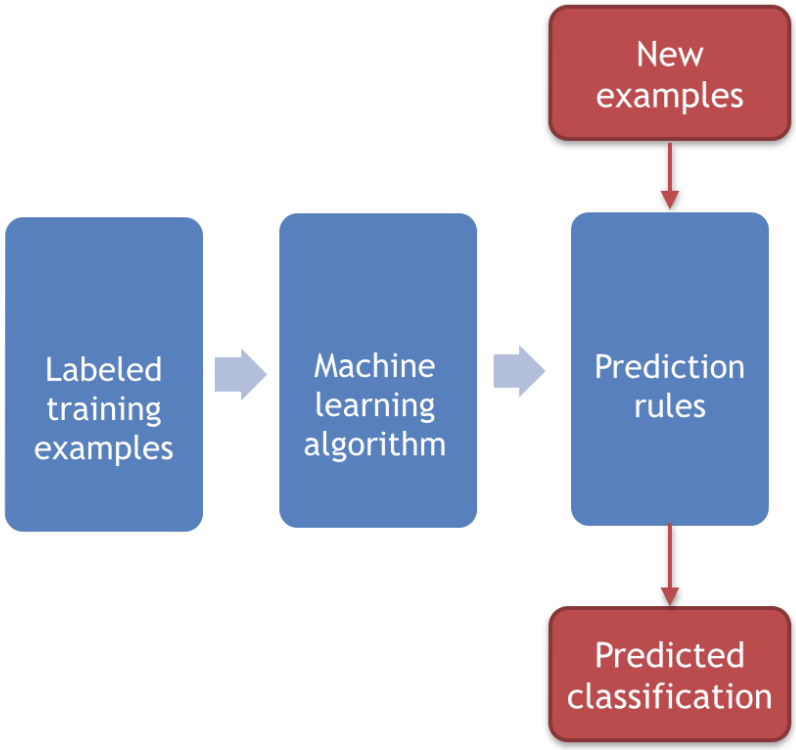

Figure 1.1 Typical supervised learning problem

Machine learning algorithms are split into 2 categories:

supervised learning Applications in which the training data comprises examples of the input vectors along with their corresponding target vectors are known as supervised learning problems.

unsupervised learning In other pattern recognition problems, the training data consists of a set of input vectors x without any corresponding target values. The goal in such unsupervised learning problems may be to discover groups of similar examples within the data: it is called clustering.

Machine learning covers these areas:

classification assign a category to each object (OCR, text classification, speech recognition)

regression predict a real value for each object (prices, stock values, economic variables, ratings)

clustering partition data into homogeneous groups (analysis of very large data sets)

ranking order objects according to some criterion (relevant web pages returned by a search engine)

dimensionaly reduction find lower-dimensional manifold preserving some proper-

ties of the data (computer vision)

density estimation learning probability distribution according to which data have been sampled

1.4 Test bench

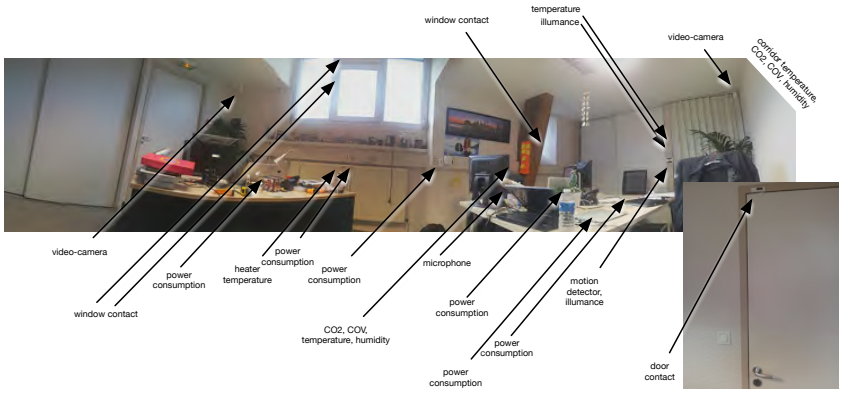

Figure 1.2 Sensor test-bed at Grenoble INP

The testbed is an office in Grenoble Institute of Technology, which accommodates a professor and 3 PhD students. The office has frequent visitors with a lot of meetings and presentations all through the week. The setup for the sensor network includes (see figure 1.2):

- 2 video cameras for recording real occupancy and activities.
- 2 luminosity sensors with different sensitivities
- 4 indoor temperature, for the office and the bordering corridor
- 2 COV+CO2 (see figure 1.3) concentration sensors for office and corridor
- 1 relative humidity sensor
- 4 door and window contact sensors
- 1 motion detector (see figure 1.5)
- 1 binaural microphone for acoustic recordings
- 5 power meters (see figure 1.4)
- outdoor temperature, nebulousness, relative humidity, wind speed and direction, . . . from weather forecasting services
- a centralized database with a web-application for retrieving raw data from different sources continuously

All the data possibly used for estimating the occupancy will be called features as in machine learning community.

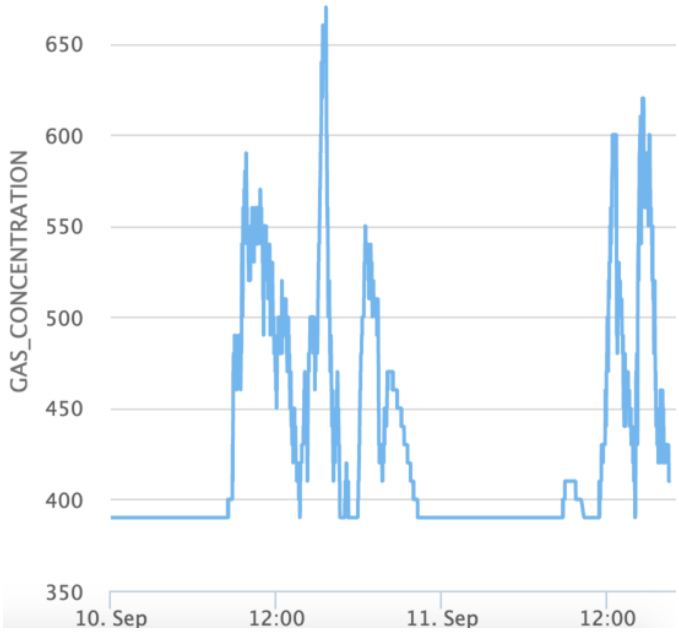

Figure 1.3 Example of raw CO2 concentration data

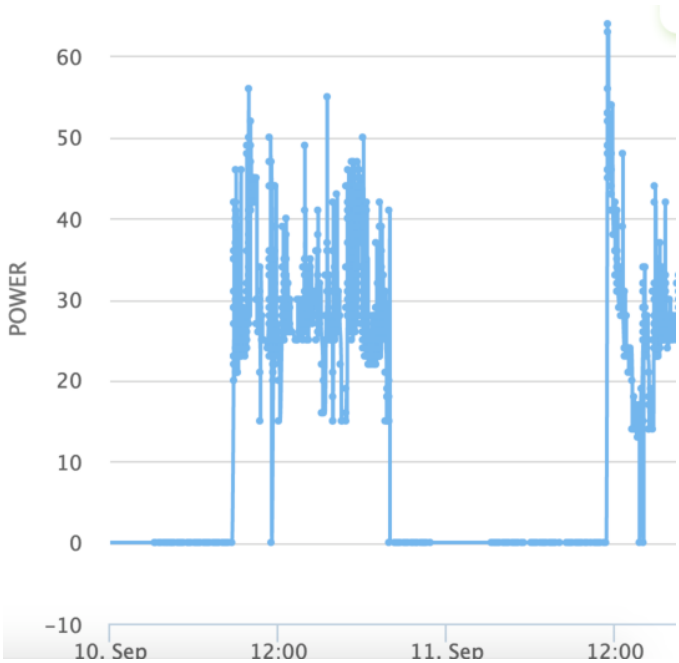

Figure 1.4 Example of raw power consumption data

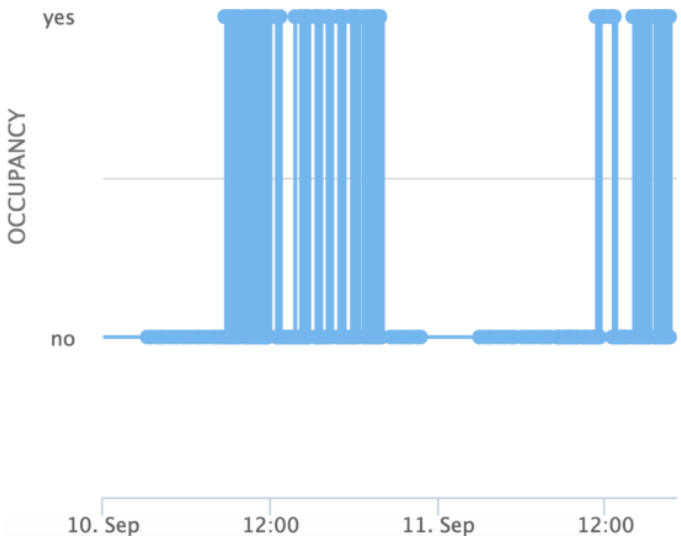

Figure 1.5 Example of raw motion detection data

The objective followed by this tutorial is to use Scientific Python to determine what are the most relevant sensors to estimate the number of occupants in the office. We are going to determine the best way to combine measurements to get a good occupancy estimator. It will be extended to activity estimation.

1.5 Machine learning with scikit-learn

The scikit learn project has started in 2007 as a Google Summer code project proposed by David Cournapeau. Later, still in 2007, Matthieu Brucher started to work on this project as a part of his PhD thesis. In 2010, Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel from INRIA took leadership of the project and made the first public release in February 1st, 2010. Since then, several releases appeared and a thriving international community is leading the development.

Scikit-learn provides a rich environment with state of the art implementations of many well known machine learning algorithms, while maintaining an easy to use interface tightly integrated with the Python language. To start with scikit-learn, visit http://scikit-learn.org/stable/index.html and click on examples. You will find all the gallery of machine learning at http://scikit-learn.org/stable/auto_examples/index.html.

Then, to go further into classification, go to the home page http://scikit-learn.org/stable/supervised_learning.html#supervised-learning, then click on decision

tree: http://scikit-learn.org/stable/modules/tree.html. To see a documentation about DecisionTreeClassifier() function, for instance, click http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.

Here is a typical code (but that cannot be executed) using scikit-learn:

```
from sklearn import tree
classifier = tree.DecisionTreeClassifier()
classifier.fit(train_data, label_data)
classifier.predict(test_data)
```

For training, scikit-learn embeds a set of testing data corresponding to iris flower species (see http://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html). The data consist of the following features in the Iris dataset: sepal length in cm, sepal width in cm and petal length in cm. Target classes to predict are the iris setosa, the iris versicolour and the virginica.

Machine learning generates models from data. For that reason, we will start by discussing how data can be represented in order to be understood by the computer.

Most machine learning algorithms implemented in scikit-learn expect data to be stored in a two-dimensional array or into a matrix. The arrays can be either numpy arrays, or in some cases, scipy.sparse matrices. The size of the array is expected to be $[n_{\text{samples}}, n_{\text{features}}]$ with:

n_{samples} the number of samples: each sample is an item to process (e.g. classify). A sample can be a document, a picture, a sound, a video, an astronomical object, a row in database or CSV file, or whatever you can describe with a fixed set of quantitative traits.

n_{features} the number of features or distinct traits that can be used to describe each item in a quantitative manner. Features are generally real-valued, but may be boolean or discrete-valued in some cases. The number of features must be fixed in advance. However it can be very high (e.g. millions of features) with most of them being zeros for a given sample. In this case, scipy sparse matrices can be useful in that they are much more memory-efficient than numpy arrays.

Whatever the classifier is, scikit-learn proposes common methods to process data:

model.fit() fit training data. For supervised learning applications, it accepts two arguments: the data X and the labels y (e.g. model.fit(X, y)). For unsupervised learning applications, it accepts only one single argument, the data X (e.g. model.fit(X)).

model.predict() given a trained model, predict the label of a new set of data. This method accepts one argument, the new data X_new (e.g. model.predict(X_new)),

and returns the learned label for each object in the array.

model.predict_proba() For classification problems, some estimators also provide this method, which returns the probability that a new observation has each categorical label. In this case, the label with the highest probability is returned by model.predict().

model.score() For classification or regression problems, most estimators implement a score method. Scores are between 0 and 1. A larger score indicating a better fit.

model.transform() Given an unsupervised model, transform new data into the new basis. It also accepts one argument X_new, and returns the new representation of the data based on the unsupervised model.

model.fit_transform() Some estimators implement this method, which performs more efficiently a fit and a transform on the same input data.

Exercise 1.1 K-means clustering

Here is an application example with a clustering algorithm named K-means:

```
from sklearn.cluster import KMeans
from sklearn import datasets
from pylab import *

iris = datasets.load_iris()
X, y = iris.data, iris.target
k_means = KMeans(n_clusters=3, random_state=0) # Fixing the RNG in kmeans
k_means.fit(X)
y_pred = k_means.predict(X)

scatter(X[:, 0], X[:, 1], c=y_pred);
show()
```

Resulting classification is illustrated in figure 1.6.

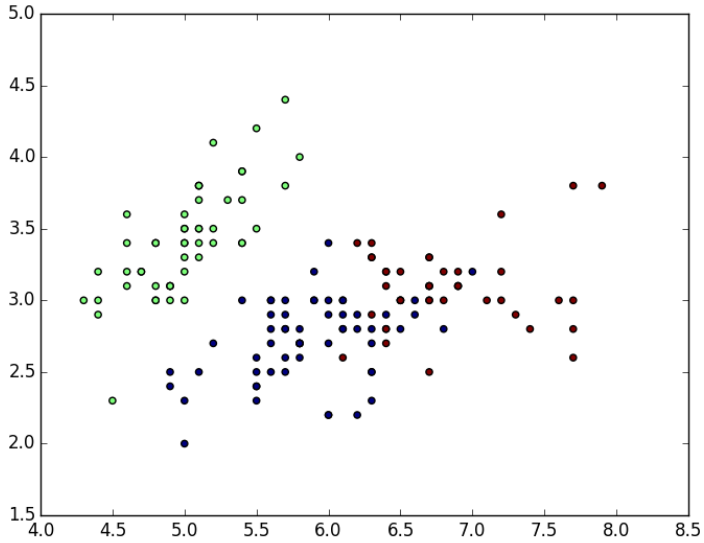

Figure 1.6 Results of the kmean clustering

Tutorials and user guide about Scikit learn can be found at http://scikit-learn.org/stable/tutorial/ and http://www.math.unipd.it/~aiolli/corsi/1213/aa/user_guide-0.12-git.pdf.

Chapter 2

Estimation of occupancy in an office

2.1 Data preprocessing

Exercise 2.1 Labelling for supervised learning (question)
How to label data with number of occupants for cross validation?

- video camera for recording real occupancy (actually average occupancy per time quantum) and activities
- keyboard connected to a Raspberry PI module used by the visitors for updating occupancy in the office when they enter and leave the office
- occupancy from other estimators used a reference such a power consumptions or from motion detections
- counting gate (but not available here)

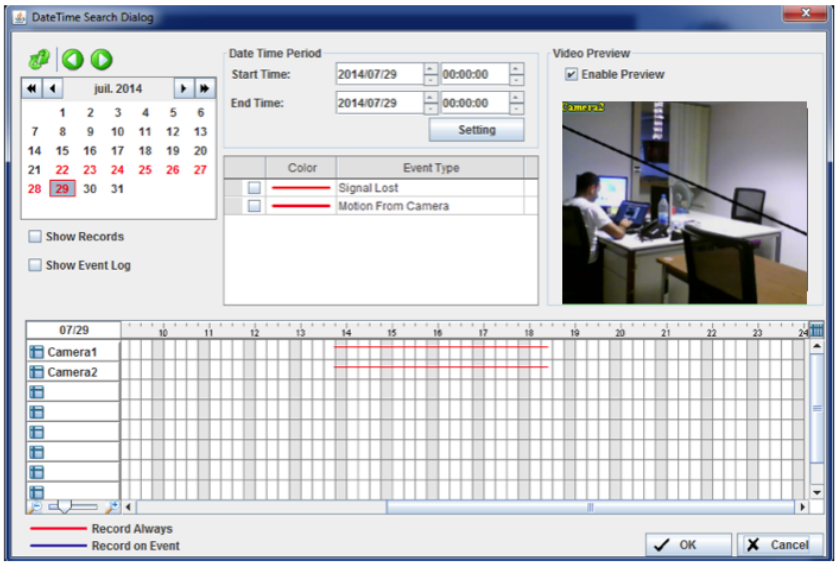

Figure 2.1 Screenshot of the video camera monitoring

2.2 Information gain to measure the purity of a decision

Entropy is an attribute of a random variable that measures its disorder. The higher the entropy is, the higher is the disorder associated with the variable i.e. the less it can be predicted. Mathematically, entropy is defined by:

$$H(y) = \sum_{i=0}^{n-1} -p(y = \mathscr{Y}_i) \log_2 p(y = \mathscr{Y}_i)$$

where:

y:	a discrete random variable (y_k is a sample) whose value domain is $\text{dom}(Y) = \{\mathscr{Y}_0, \ldots, \mathscr{Y}_{n-1}\}$
$H(y)$:	entropy of a random variable y
$p(y = \mathscr{Y}_i)$:	probability for y to be equal to the value $\mathscr{Y}_i$

Exercise 2.2 Concept of entropy(run example1.py)

Compute the entropy of the following variables:

- $X1 = [0,0,1,1,0,1,0,0,1,1,0,1]$
- $X2 = [1,1,1,1,1,1,1,1,1,0,1,1]$
- $X3 = [1,1,1,1,1,1,1,1,1,1,1,1]$

Solution:

```
import math

X1 = [0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1]
```

```
X2 = [1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1]
X3 = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

def entropy(variable_data): #occupancy is the target
    count0, count1 = 0, 0  # counter for the class0 and class1 in the occupancy column
    for i in range(len(variable_data)):
        if variable_data[i] == 0: count0 += 1
        else:  count1 += 1
    p0, p1 = count0 / len(variable_data), count1 / len(variable_data)
    v0 = math.log(p0, 2) if p0 != 0 else 0  # to avoid log(0)
    v1 = math.log(p1, 2) if p1 != 0 else 0  # to avoid log(0)
    return - p0 * v0 - p1 * v1

print(entropy(X1))  # 1.0
print(entropy(X2))  # 0.41381685030363374
print(entropy(X3))  # 0.0
```

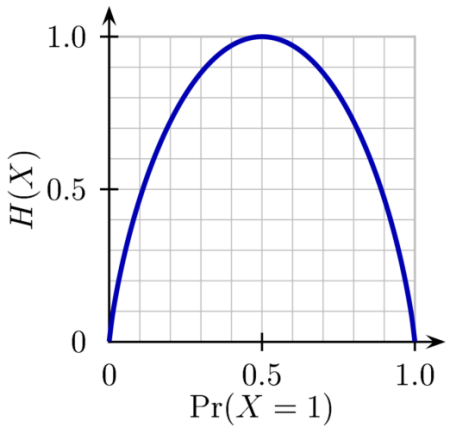

Figure 2.2 Entropy vs. probability
If probability of an event is 50%, it corresponds to the highest disorder i.e. entropy=1

Information gain can now be defined for two random variables, x and y as:

$$IG(x, y) = H(y) - H(y|x) \tag{2.1}$$

where:

x:	an input discrete random variable (x_k is a sample) used as feature ($\text{dom}(x) = \{\mathscr{X}_0, \ldots, \mathscr{X}_{n_{\mathscr{X}-1}}\}$)
y:	the target discrete random variable (y_k is a sample) ($\text{dom}(y) = \{\mathscr{Y}_0, \ldots, \mathscr{Y}_{n_{\mathscr{Y}-1}}\}$)
$H(y)$:	the entropy of y
$H(y\|x)$:	the conditional entropy of y given x with:

$$H(y|x) = \sum_{i=0}^{n_{\mathscr{X}-1}} p\left(x = \mathscr{X}_i\right) H\left(y_{\{k|x_k=\mathscr{X}_i\}}\right) \tag{2.2}$$

The higher the reduction of disorder according to feature x is, the more is the information gained for determining y thus making x a good feature to use for classifying y.

Exercise 2.3 Information gain (run example2.py)
In the file 'data.txt', we have 2 features: door and window position, and one target: occupancy. Which one of the features (door, window) is the most important? Calculate the entropy and the information gain for the door and window values.

Solution:
1-First step: read the values of data in the file 'data.txt'

```
door, window, occupancy_data = [], [], []
file = open("data.txt", 'r')
Title= file.readline()
for line in file:
 line_token = line.split(";")
 door.append(float(line_token[0])), window.append(float(line_token[1])), →
     ↪ occupancy_data.append(float(line_token[2])) # store the values of each column →
     ↪ in example2.txt in 3 separate lists
file.close()

print(occupancy_data)
```

2-Last step: calculate the information gain

```
def information_gain(feature_data, occupancy_data):
 occupancy_data0, occupancy_data1, feature_data0, feature_data1 = [], [], [], []
 for i in range(len(occupancy_data)):
   if feature_data[i] == 0:
     feature_data0.append(feature_data[i])
     occupancy_data0.append(occupancy_data[i])
   else:
     feature_data1.append(feature_data[i])
     occupancy_data1.append(occupancy_data[i])
 entropy0, entropy1 = entropy(occupancy_data0), entropy(occupancy_data1)
 return entropy(occupancy_data) − (len(feature_data0) / len(occupancy_data)) ∗ →
     ↪ entropy0 − (len(feature_data1) / len(occupancy_data)) ∗ entropy1
```

To go further using real data, two datasets are available: a training dataset in SQLite database 'office.db' and a validation dataset is 'testing.db'. Training data covers for 11 days from 04-May-2015 to 14-May-2015 while validation data is collected over 4 days from 17-May-2015 to 21-May-2015.

Exercise 2.4 Reading from a SQLite database(use SQLite browser)
Using 'sqlitebrowser' for instance, open the databases 'testing.db' and 'office.db' with a SQLite browser.
The beginning of 'example3.py' reads data from the databases and processes them.

Solution: There are three kinds of data: continuous data (figure 2.3), discrete data (figure 2.4) and occupancy labels (figure 2.5).

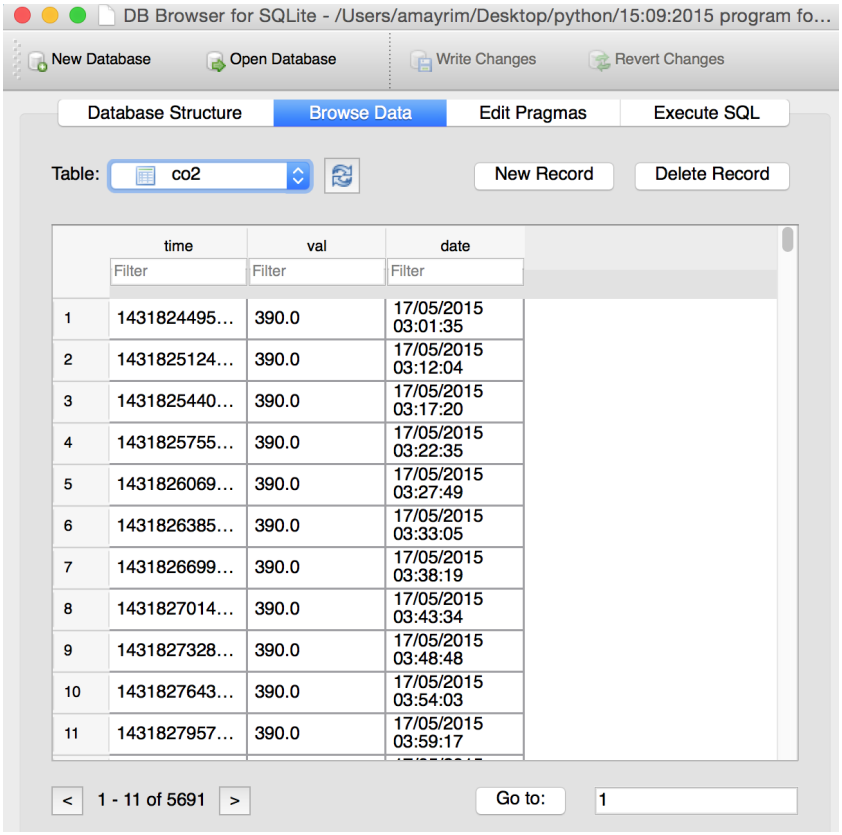

Figure 2.3 CO2 concentration

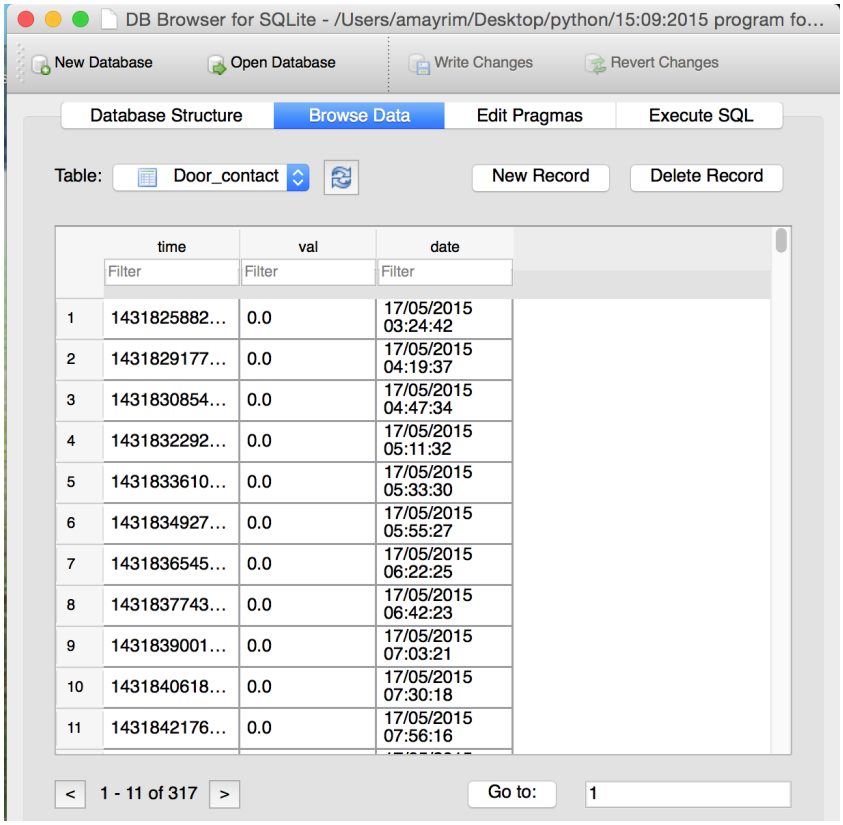

Figure 2.4 Door position

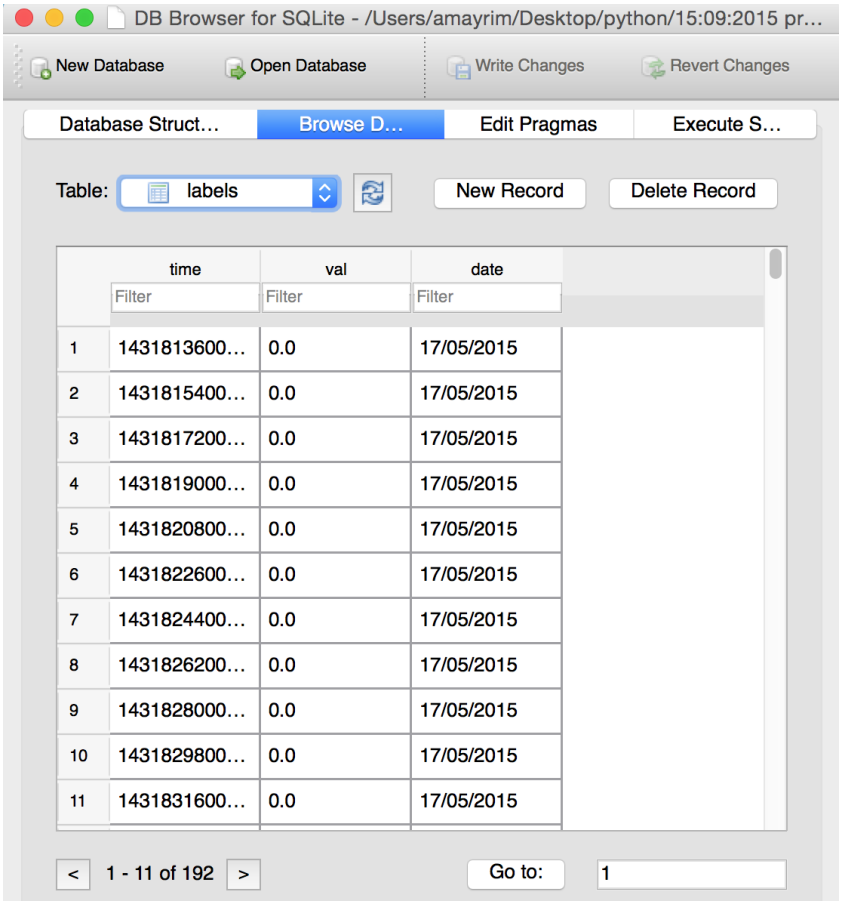

Figure 2.5 Occupancy

Another issue has to be solved: how to calculate the information gain for continuous variables such as CO2 concentration or (average) occupancy.

To mathematically calculate the information gain for continuous data, it is necessary to discretize the features, which have values that are continuous in nature. A typical discretization function splits a large continuous range into several sub-ranges. However, such a function relies upon:

- sorting the values of the feature to be discretized.
- determining a cut-point for splitting, according to some criteria (maximum and minimum value for each feature).

Examples of considering the prior discretization:

1. CO_2 (in ppm):

Disc	$\{[390,450], [450,690], [690,900], [900,1300], [>1300]\}$

2. motion fluctuations (counting of occupants presence during a time quantum):

Disc	{[0,2], [2,4], [4,6], [6,9], [> 9]}

Exercise 2.5 Prior discretization of continuous variables
Run 'example3.py' and observe the results, propose another discretization and compare the results.

Solution: According to the prior discretization, it comes out that the best sensors to estimate occupancy are :

Feature	IG_1
microphone	0.68
motion fluctuations	0.62
occupancy from physical model	0.55
power consumption	0.5
CO_2 mean	0.5
CO_2 derivative	0.452
door's position	0.41
window's position	0.341
indoor temperature	0.08
mean ($temp_{outside}$-$temp_{inside}$)	0.07
day type	0

Finally, after removing less important features, the main informative features are found to be:

1. Microphone.
2. Occupancy estimation from CO_2 physical model.
3. Motion detector counting.
4. Occupancy estimation from power consumption.
5. Door position.

2.3 Decision tree as a classifier

Decision trees are commonly used in learning algorithms because it yields to a sequence of intermediate decisions that lead to a final decision. Such classifiers provide very good results but in addition, the results are easy to read, analyze and adapt. A decision tree selects a class by descending a tree of decision nodes. Each internal node represents a comparison of a single feature value with a learned threshold. Therefore, data are split at each node in a tree according to a decision rules, which corresponds to nested if-then-else rules. The target of a decision tree is to select features that are the most useful for classification. One quantitative measurement of the usefulness of a feature is information gain. We can distinguish a root node, internal nodes and leaf or terminal nodes corresponding to a class label.

Exercise 2.6 Generation of a decision tree (run example4.py)

Consider the following data:

```
X = [[0, 0], [1, 1],[0, 1]]  # training data (2 features, 3 samples)
Y = [0, 1, 0]  # target data (3 samples)
```

Propose a classifier based on decision tree using scikit learn and classify the sample Xnew = [2, 2]

Solution:

```
from sklearn import tree
X = [ [0, 0], [1, 1], [0, 1] ]  # training data (2 features 3 samples)
Y = [0, 1, 0]  # labelling data
classifier = tree.DecisionTreeClassifier(random_state=0, criterion='entropy', →
    ↪ max_depth=None)
classifier.fit(X, Y)  # fit the tree according to training and labelling data
print(classifier.predict([[2, 2]]))  # predict the new target sample (validation)
with open("tree.dot", 'w') as file:
    f = tree.export_graphviz(classifier, out_file=file) # save the tree as dot file and open →
    ↪ it using Graphviz or a text editor (at lear WordPad on Microsoft Windows)
```

Figure 2.6 represents the learned decision tree. Right arrow corresponds to the decision when condition in the box is satisfied.

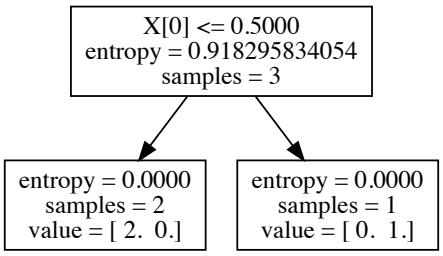

Figure 2.6 A decision tree computed by C45 algorithm

To go further, we are going to use scikit-learn to generate the decision tree for estimating the number of occupants based on datasets ‘office.db’ and ‘testing.db’.

Exercise 2.7 Depth of decision trees Run ‘example5.py’ and observe the results, then propose some changes in DecisionTreeClassifier function to limit the depth of the tree.

The C4.5 decision tree algorithm has been used to perform recognition by using aggregated features and the labels extracted from video cameras. 5 occupancy levels have been defined to generate decision trees because of the maximum number of

occupants in the office.

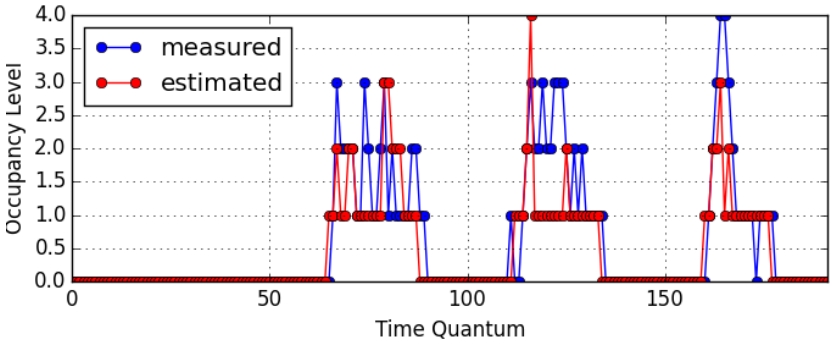

Figure 2.7 Occupancy estimation from DT using all features

Figure 2.7 shows the results obtained from the learned decision tree considering all the features as input to the detection model, where we plot both actual occupancy profile and the estimated profile as a graph of number of occupants with respect to time (quantum time is 30 minutes). The accuracy achieved is 79% (number of correctly estimated points divided by the total number of points), and average error 0.32 (average distance between actual points and estimated points). The following table represents the values of average error for each class of estimation:

	Average error	support
class 1	0.015	132
class 2	0.67	22
class 3	0.52	23
class 4	1.5	11
class 5	2.6	4
avg/total	0.3	192

Figure 2.8 shows the result obtained from the decision tree considering only the main features. It leads to improvement in occupancy estimation with an accuracy of 82% and an average error of 0.24 occupant in average. Additionally, the results indicate that microphone, occupancy estimation from CO_2 physical model, motion detector, power consumption and door contact have the largest correlation with the number of occupancy detection.

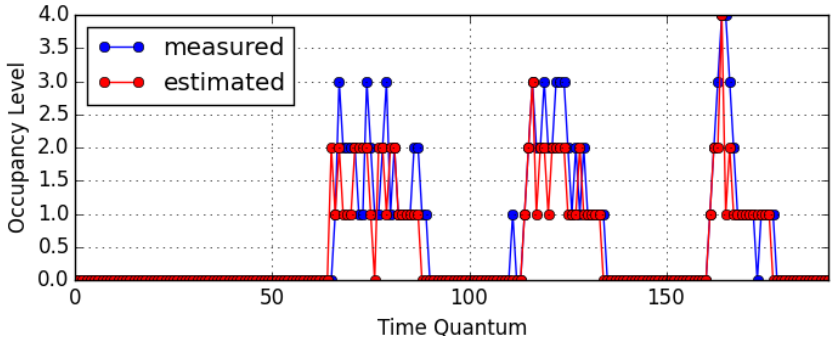

Figure 2.8 Occupancy estimation from DT using main features

Here are the errors for each class when considering all the features:

	Average error	support
class 1	0.02	132
class 2	0.5	22
class 3	0.4	23
class 4	1	11
class 5	2	4
avg/total	0.24	192

Comparing the results obtained from the decision tree using all the features with the one using only the main features, a significant improvement in occupancy estimation for all levels can be observed. The estimation of human activities is proposed also. Figure 2.9 represents the results of estimation human activities with average error=0.02. The considered activities are given by:

- Professor working on computer (1)
- Professor participating to a physical meeting (2)
- Professor participating to a virtual meeting (phone call, Skype) (3)

The considered features are:

- Estimation of occupancy level
- Professor desk consumption
- Professor laptop consumption

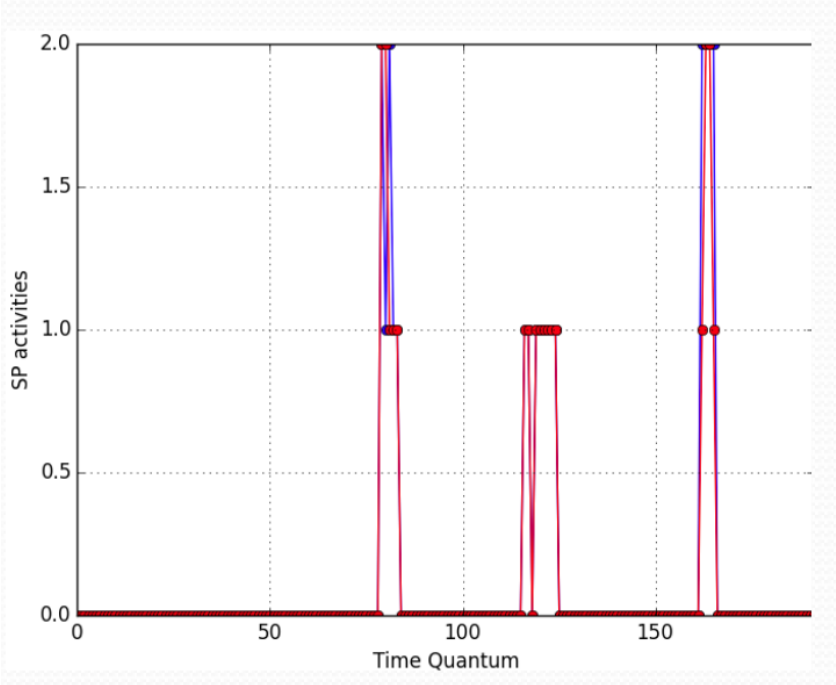

Figure 2.9 Activities estimation using decision tree

2.4 Assessing the result quality

Exercise 2.8 Performance assessment (run example6.py)
Generally, in machine learning, the validation of estimated results can be checked by considering F-score, while in our estimation we considered the average error.so a question arise :
Why average error is more interesting than F-score?

Solution: a-Let's see some important definitions

- Precision: fraction of elements correctly classified as positive out of all the elements the algorithm classified as positive $\frac{t_p}{(t_p+f_p)}$
- recall: fraction of elements correctly classified as positive out of all the positive elements $\frac{t_p}{(t_p+f_n)}$
- F-score: the harmonic mean of precision and recall $2 * \frac{(precision*recall)}{(precision+recall)}$. The best score possible for a class is 1 and worst is 0.
- Average error: average distance between actual points and estimated points.

where:

t_p:	true positive i.e. element predicted to be in class i is really in class i
f_p:	false positive i.e. element predicted to be in class i is really not in class i
f_n:	false negatives i.e. element predicted not to be in class i, but is really in class i

b-Let's compute the accuracy and average error to the first estimated points and

second estimated points in the file 'error.txt'
1- Read the data in the 'data.txt'

```
# First step: we are going to read the data in the (data.txt)
import numpy
actual_points,estimated1,estimated2=[],[],[]
file = open("data1.txt", 'r')
Title = file.readline()# read the first line of variable's names
for line in file:
    line_tokens = line.split(";")
    actual_points.append(float(line_tokens[0])), estimated1.append(float(line_tokens[1])), →
        ↪ estimated2.append(float(line_tokens[2]))
file.close()
```

2- Calculate accuracy and average error

```
error,count =[],0
for i in range(len(actual_points)):
    distance = actual_points[i]−estimated1[i]
    if distance == 0: count += 1 # count the correctly estimated points to calculate the →
        ↪ accuracy
    error.append(abs(distance))
accuracy = count ∗ 100 / len(actual_points)  # accuracy %
average_error = (numpy.sum(error)) / len(actual_points)
print("average error = ", average_error)
print("accuracy=", accuracy, "%")
```

Observe the results:

	Accuracy	Average error
First estimated points	0.33	0.83
Second estimated points	0.33	0.62

Note that average error is taking into account each change in the estimation values, while the accuracy consider only the correctly estimated points, even if there are changing in the other estimation values.

2.5 Random forest as a robust solution

A random forest is a meta-estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. In other words, random forests is a learning method for classification and regression that operate by constructing a lot of decision trees at training time and outputting the class that is the mode of the classes output by individual trees (see http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html and http://scikit-learn.org/stable/modules/ensemble.html).

Exercise 2.9 Occupancy estimator using random forest (run example7.py)
Using random forest will allow you to test several numbers of classifier, each tree is constructed using a different bootstrap sample from the original data. About one-third of the cases are left out of the bootstrap sample and not used in the construction of the current tree. However, we are going to apply the random forest algorithm and compare the results with the decision tree. It will provide more validation to our estimations of occupancy.

Consider the following data:
X = [[0, 0], [1, 1],[0, 1]] # training data (2 features, 3 samples)
Y = [0, 1, 0] # target data (3 samples)

Propose a classifier based on random forest using scikit learn and classify the sample Xnew = [2, 2]

Solution:

```
from sklearn.ensemble import RandomForestClassifier
X = [ [0, 0], [1, 1], [0, 1] ] # training data (2 features 3 samples)
Y = [0, 1, 0] # labelling data
classifier = RandomForestClassifier(n_estimators=10, random_state=0, criterion='
    ↪ entropy',max_depth=None)#n_estimators is the number of trees in the
    ↪ forest
classifier.fit(X, Y) # fitting the tree according to training and labelling data
print(classifier.predict([[2, 2]])) # predict the target (validation)
```

Run ‘example8.py’ and compare the current results of random forest with the the results of example5.

Chapter 3

appendix

3.1 Basic Python

The Python programming language is one of the most popular programming languages for scientific computing. Considering the number of papers that compare it to Matlab, we can infer a growing popularity in scientific community. Thanks to its high level interactive nature and its maturing ecosystem of scientific libraries, it is an appealing choice for algorithmic development and exploratory data analysis. Yet, as a general purpose language, it is increasingly used not only in academic settings but also in industry.

- Python is interpreted, object-oriented, high level with automatic memory management
- Python is free, cross-platform and open source, whereas Matlab is a closed-source commercial product
- Python codes are compiled to executable during the first interpretation increasing the running speed
- Python is a multi-purpose programming language than be used both for fast prototyping codes and for the development of complete applications (see https://www.python.org/about/success/)
- Python integrates well with other languages such as C or C++.
- Python includes natively an impressive number of general-purpose or more specialized libraries, and yet more external libraries are being developed by Python enthusiasts.
- Python is known as one of the easiest language to learn (see for instance http://jasonrbriggs.com/python-for-kids/)
- Python code is laconic: a lot can be done in few lines. The Python 'hello world' is: `print('Hello World!')`

Python 2 is not fully compatible with Python 3. Python 2 will not be released any more. Python 3 is a better choice for beginners.

Working with Python can be done just like with the command line window of Matlab using IDLE or iPython with a lot more facilities. For beginners that would like to test Python, we recommend the Pyzo distribution because it is easy to install on any OS but also because it installs iPython, scientific libraries and a basic integrated development environment named IEP (http://www.pyzo.org/downloads.html). Nevertheless, a simple text editor such as 'Sublime Text' or 'Komodo Edit' can also be used to develop and run codes. For those who would like an advanced integrated development environment, we recommend PyCharm (http://www.jetbrains.com/pycharm/)

Getting help about an function or an object can be done from within Python. For instance, to get help about a function 'list', you can do: help(list), print(list.__doc__) or dir(list). Official documentation can be found at http://docs.python.org/3/.

3.1.1 Data types

In Python, numbers can be:

integers (int) with an unlimited precision: 1234, -24, 0
floating numbers (float) with an unlimited precision: 1.23, 3.14e-10, 4E210, 4.0e+210
octal or hexadecimal numbers : 0177, 0x9ff
complex 3+4j, 3.0+4.0j, 3J

Other Python main data types are:

strings (str) : 'spam', "guido's"
immutable table (tuple) (1, 'spam', 4, 'U')
dynamic tables (list) : [1, [2, 'three'], 4]
dictionnaries (dict) : 'food': 'spam', 'taste': 'yum'
files : text = open('eggs', 'r').read()

Here are some example of codes illustrating basic data types and how they can be processed:

```
print(7/3)  # print is not required in iPython
print(7.5/3)
i=3, j=2
print(j**2+j**i)  # power is represented by `**'
from math import *
print(cos(exp(-9)*pi+4+5j))
myvar = 3  # assignment (this is a comment)
```

```
myvar += 2
print(myvar)  # 5
myvar −= 1 # no semi−colon at the end
print(myvar)  # 4
# a variable is not explicit typed. myvar was refering to an integer. It will now refer to a character →
    ↪ string.
myvar = """This is a multiline comment.
The following lines concatenate the two strings."""  # here is a string including a carriage return
mystring = "Hello"
mystring += "  world."  # use `+' to concatenate strings
print(mystring)  # Hello world.
print(mystring[2:5])  # llo
print(mystring[2:−4])  # llo wo (negative index count from the end)
print('a %s parrot' % 'dead')  # a dead parrot
[x for x in mystring]  # ['H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '.']
'world' in mystring  # True
myvar, mystring = mystring, myvar # This swaps the variables in one line. It does not violate  →
    ↪ strong typing because values are not actually being assigned, but new objects are bound  →
    ↪ to the old names.
```

Let's now examine immutable tuples and dynamic lists:

```
a=(1,2,3)  # a tuple
a[1]  # 2
a[1] = 3  # error: 'tuple' object does not support item assignment
mylist = ["List item 1", 2, 3.14]  # heterogeneous list
print(mylist[:])  # ['List item 1', 2, 3.1400000000000001]
print(mylist[0:2])  # ['List item 1', 2]
print(mylist[−3:−1])  # ['List item 1', 2]
print(mylist[1:])  # [2, 3.14]
mylist.extend([4,5,6])  # ['List item 1', 2, 3.14, 4, 5, 6]
mylist.append(7)  # ['List item 1', 2, 3.14, 4, 5, 6, 7]
print(mylist[::2])  # ['List item 1', 3.14, 5, 7]
mylist[0]=8  # [8, 2, 3.14, 4, 5, 6, 7]
mylist.sort() # mylist = [2, 3.14, 4, 5, 6, 7, 8]
mylist.reverse()  # [8, 7, 6, 5, 4, 3.14, 2]
mylist.index(3.14)  # 5
del mylist[2:4]  # [8, 7, 4, 3.14, 2]
mylist[1:2] = [4.5,2.3]  # [8, 4.5, 2.3, 4, 3.14, 2]
[x for x in range(2,23,3)]  # [2, 5, 8, 11, 14, 17, 20]
[x for x in range(2,23,3) if x%2 == 1]  # [5, 11, 17]
[x * y for x in [1, 2, 3] for y in [3, 4, 5]]  # [3, 4, 5, 6, 8, 10, 9, 12, 15]
any([i % 3 for i in [3, 3, 4, 4, 3]])  # "any" returns true if any item in the list is true
sum(1 for i in [3, 3, 4, 4, 3] if i == 4)  # 2
```

Dictionaries are also common:

```
d1 = {}
d2 = {'spam': 2, 'eggs': 3}
d3 = {'food': {'ham': 1, 'egg': 2}}
d2['eggs']  #[Out]# 3
```

```
d3['food']['ham'] #[Out]# 1
'eggs' in d2 #[Out]# True
d2.keys() #[Out]# dict_keys(['spam', 'eggs'])
d2.values() #[Out]# dict_values([2, 3])
len(d1) #[Out]# 0
d2[5] = 'ok' #[Out]# {'spam': 2, 'eggs': 3, 5: 'ok'}
del d2[5] #[Out]# {'eggs': 3, 'spam': 2}
d2.items() #[Out]# dict_items([('spam', 2), ('eggs', 3)])
for k, v in d2.items():          # key: spam val: 2
  print('key: ', k, 'val: ', v)    # key: eggs val: 3
```

Comparison operators are similar to other languages (==, !=, <, <=, >, >=). Additionally, Python offers 'is' and 'is not' comparison operator to test object identities i.e. addresses is memory:

```
L1 = [1, ('a', 3)]
L2 = [1, ('a', 3)]
L1 == L2, L1 is L2 #[Out]# (True, False)
```

Built-in types are detail at https://docs.python.org/3/library/stdtypes.html. More advanced data types are presented at https://docs.python.org/3/library/datatypes.html.

'global' keyword gives access to variable of the module scope:

```
number = 5
def f():
  print(number)

def g():
  print(number)
  numner = 3

def h():
  global number
  print(number)
  number = 3

f() # 5
g() # 5
g() # 5
h() # 5
h() # 3
```

3.1.2 Control flows

Python control flows look like those of other languages except curly brackets are not used but indentations determine the code blocks:

```
x = -3
if x < 0:
   x = 0
   print('Negative change to zero')
elif x == 0:
   print('Zero')
elif x == 1:
   print('Single')
else:
   print('None')  # Negative change to zero

for item in [12, 'test', 0.1+1.2J]:
   print(item)  # 12, test, (0.1+1.2j)

for i in range(3,10,2):
   print(i)  # 3, 5, 7, 9

for number in range(10):
   if number in (3, 4, 7, 9):  # Check if number is one of the numbers in the tuple
      break  # "Break" terminates a for without executing the "else" clause
   else:
      continue  # "continue" starts the next iteration of the loop. It's useless here, as it's the last →
      ↪ statement of the loop.
else:  # The "else" clause related to for loop is optional and is executed only if the loop didn't " →
      ↪ break"
   pass  # Do nothing

i=0
rangelist = range(10)
while rangelist[i] < 4:
   print(rangelist[i])  # 0, 1, 2, 3
   i += 1
```

To learn more about control flows, see https://docs.python.org/3/tutorial/controlflow.html.

3.1.3 Functions

Functions can return several heterogeneous values, admit a variable number of arguments with default values:

```
def ask_ok(prompt,retries=4,complaint='Yes or no, please'):
   while True:
```

```
        ok = input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        elif ok in ('n', 'no'):
            return False
        retries = retries - 1
        if retries < 0:
            print(complaint)
            raise IOError('refused')

ask_ok('Give answer: ', 2, complaint='Reply yes or no')
# Give answer: yeah
# Give answer: not
# Give answer: none
# Reply yes or no
# →
    ↪ ----------------------------------------------------------------------
    ↪
# OSError...

def passing_example(a_list, an_int=2, a_string="A default string"):
    a_list.append("A new item")
    an_int = 4
    return a_list, an_int, a_string

passing_example([1, 2, 3], 10) ## ([1, 2, 3, 'A new item'], 4, 'A default string')

def f(x):
    return x % 2 != 0 and x % 3 != 0

for element in filter(f, range(2, 25)):
    print(element, end='  ')
print()  # 5 7 11 13 17 19 23

def g(x):
    return x**3

for element in map(g, [2, 4, 6, 8]):
    print(element, end='  ')
print()  # 8 64 216 512

funcvar = lambda x: x + 1  # lambda function; same as def funcvar(x): return x + 1
print('0)', funcvar(1))

def vargtest(a, b, *nkw, **kw):
    """
    display regular args and variable args
    """
    print('a is: ', a)
    print('b is: ', b)
    for each_nkw in nkw:
        print('additional non-keyword arg:', each_nkw)
    for each_kw in kw:
        print('additional keyword arg: "%s": %s' % (each_kw, kw[each_kw]))
```

```
vargtest(1,2,3,4,x=2,y=3)
vargtest(1,2,*(3,4),**{'x':2,'y':3})
# both yield
# a is: 1
# b is: 2
# additional non-keyword arg: 3
# additional non-keyword arg: 4
# additional keyword arg: "y": 3
# additional keyword arg: "x": 2
```

To learn more about functions, see https://docs.python.org/3/reference/compound_stmts.html#function-definitions.

3.1.4 Object oriented programing

Python can also be used to developed object oriented codes. It supports multiple inheritance and overloading but does not support method polymorphism. In practice, it is not a big issue because of the default argument values.

Here are some code examples:

```
class Dog:
    counter = 0
    def __init__(self, name):  # constructor
        self.name = name
        Dog.counter += 1
    def bark(self):
        return self.name + '(' + str(Dog.counter) + ') wouaf'

dog1 = Dog('Lassie')
dog2 = Dog('Rex')
print(dog1.bark())  # Lassie(2) wouaf
print(dog2.bark())  # Rex(2) wouaf


class Vehicle:
    def __init__(self, initial_covered_distance):
        self.covered_distance = initial_covered_distance
    def add_distance(self, new_covered_distance):
        self.covered_distance += new_covered_distance

class Plane(Vehicle):  # inheritance from Vehicle
    def __init__(self, number_of_engines, initial_covered_distance):
        Vehicle.__init__(self, initial_covered_distance)  # constructor of the parent has to be called
        self.number_of_engines = number_of_engines
    def __str__(self):  # called by default by print
        return "Plane with %i engines. Covered distance is %d" % (self. →
        ↪ number_of_engines,self.covered_distance)
```

```
print(Plane(4, 150000))  # Plane with 4 engines. Covered distance is 150000

class Surface:
    def get_surface(self):
        raise NotImplementedError()
    def get_perimeter(self):
        raise NotImplementedError()
    def __str__(self):
        return 'Surface is: %f and perimeter: %f' % (self.get_surface(), self. →
        ↪ get_perimeter())

class Color:
    def __init__(self, color: int=0):
        self.color = color
    def set(self, color: int):
        self.color = color
    def get(self):
        return self.color

class Rectangle(Surface, Color):  # multiple inheritance
    def __init__(self, width: float, height: float, color: int=0):
        Color.__init__(self, color)  # all the parent constructors should be called
        Surface.__init__(self)  # all the parent constructors should be called
        self.width = width
        self.height = height
    def get_surface(self):
        return self.width * self.height
    def get_perimeter(self):
        return 2 * (self.width + self.height)

class Square(Rectangle):
    def __init(self, length: float, color: int=0):
        Rectangle.__init__(self, length, length, color)

import math

class Disk(Surface, Color):
    def __init__(self, radius: float, color: int=0):
        Color.__init__(self, color)
        Surface.__init__(self)
        self.radius = radius
    def get_surface(self):
        return math.pi * self.radius ** 2
    def get_perimeter(self):
        return 2 * math.pi * self.radius

print(Square(4.5, 3))  # Surface is: 13.500000 and perimeter: 15.000000
print(Disk(3.2))  # Surface is: 32.169909 and perimeter: 20.106193
```

3.1.5 Exceptions

Here are code samples to illustrate the syntax:

```
def some_function():
    try:
        10 / 0  # Division by zero raises an exception
    except ZeroDivisionError:
        print("Oops, invalid.")
    else: # Exception didn't occur, we're good.
        pass
    finally:
            # This is executed after the code block is run and all exceptions have been handled, even →
        ↪ if a new exception is raised while handling.
        print("We're done with that.")

some_function()
# Oops, invalid.
# We're done with that.

class Surface:
    def get_surface(self):
        raise NotImplementedError()
    def get_perimeter(self):
        raise NotImplementedError()
    def __str__(self):
        return 'Surface is: %f and perimeter: %f' % (self.get_surface(), self. →
        ↪ get_perimeter())
```

Common exceptions are NameError (attempt to access an undeclared variable), ZeroDivisionError, SyntaxError, IndexError, KeyError, IOError, AttributeError (attempt to access an unknown object attribute) and NotImplementedError to create Java equivalence to abstract classes or interfaces.

3.1.6 Working with modules

In Python, each file is a module where all the codes inside can be imported using a single import mymodule to load the content of the file. If you want to put some codes not to be run when importing, create a mean function starting with:

```
if __name__ == '__main__':
    pass  # put here the code to be execute when module is directly called and not in case of →
        ↪ importation.
```

Consider a file 'myio.py' that contains a function called 'load()'. To use that function from another module, just do:

```
import myio
myio.load()
```

All the codes in 'myio.py' is available in the myio namespace.

In order to import into the current namespace, just write:

```
from myio import *
load()
```

But be careful with multiple definitions. Just specific parts of a module can also be imported:

```
from myio import load
load()
```

Here is a small example:

```
import random
from time import clock
r = random.randint(1, 100)
print(r)  # 64
print(clock())  # 0.049354
```

In order to make visible only a subset of the entities (functions, classes,...) in a module, just declare a `'__all__'` variable as __all__ = [`'myfunction1'`, `'myfunction2 '`,... `'Class1'`, `'Class2,'`...]. The ones that are not in `'__all__'` are hidden to other modules.

Finally, all the modules in a folder can be imported at once by importing the folder. If only some modules must be imported, just create a file named `'__init__.py'` with inside a `'__all__'` variable as __all__ = [`'module1'`, `'module2'`,...].

Here are the main basic modules in Python distribution:

os file and process operations
os.path platform-independent path and filename utilities
time dates and times related functions
string commonly used string operations
math math operations and constants
cmath same than math but for complex numbers
re regulat expressions
sys access to interpreter variables

gc control over garbage collector
copy in order to copy objects

The Python basic API is very rich (see https://docs.python.org/3/library/index.html). It can be extended using for instance the 'pip' utilities (see https://pip.readthedocs.org/en/stable/) from command line:

pip list -o display the outdated Python module
pip install wheel install a utility to deploy auto-installable modules with '.whl' extension
pip install new_module.whl install the auto-installable module 'new_module' providing 'new_module.whl' is in the current folder
pip install PySide -U install or update the module named 'PySide'

3.1.7 File management

Managing text files is very easy in python:

```
 file = open('data.txt','w')
file.write('line1\n') #[Out]# 6
file.write('line2') #[Out]# 5
file.write('line3') #[Out]# 5
sequence = ['line4\n', 'line5\n']
file.writelines(sequence)
file.close()
# content of data.txt
# line1
# line2line3line4
# line5

file = open('data.txt', 'r')
print(file.read())
# line1
# line2line3line4
# line5
file.seek(0) # return to 1st byte
print(file.read(20))
# line1
# line2line3line
print(file.readline())
# 4
print(file.readline())
# line5
file.seek(0)
print(file.readlines())
# ['line1\n', 'line2line3line4\n', 'line5\n']
file.close()

file=open('data.txt','r')
```

```
for line in file:
    print(line)
#line1
#
#line2line3line4
#
#line5
#
file.close()

with open('data.txt') as file:
    for line in file:
        print(line)
#line1
#
#line2line3line4
#
#line5
#
```

Binary files can also be handled:

```
import pickle
mylist = ['This', 'is', 4, 13327]
file = open('data.dat','wb')
pickle.dump(mylist, file)
file.close()

file = open('data.dat', 'rb')
loaded_list = pickle.load(file)
file.close()
print(loaded_list)  # ['This', 'is', 4, 13327]
```

3.2 Scientific Python for matrix calculation with numpy

Because Python is very relevant for prototyping solution but also because it is open and free, the Python language is today widely used by diverse scientific communities. It is sometime used as a Matlab replacement. Matlab is more focused on matrix computation, signal processing and automatic control whereas Scientific Python is larger because it also gathers the community interested in computer science. Scientific Python is composed of a lot of libraries coming from different scientific communities, for instance:

numpy for matrix computation
scipy for integration of differential equations, optimization, interpolations
matplotlib for plotting

iPython for enhanced interactive console
sympy for symbolic calculations
pandas for data structures and analysis
scikit-learn for machine learning
pymc for stochastic calculation
libpgm for Bayesian networks

The 6 first libraries come usually together under the name Scientific Python or SciPy (pronounced 'Sigh Pie'). Documentation of the 6 libraries is available at http://www.scipy.org/docs.html. There is a magic import `from pylab import *` coming with matplotlib together with numpy ans scipy that performs proper imports in such a way that Python becomes very close to Matlab.

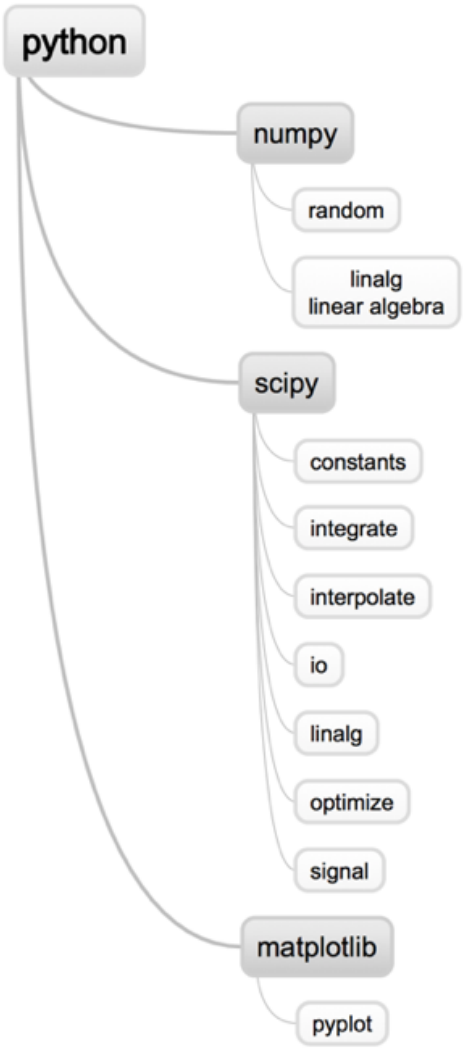

Figure 3.1 Main modules in core Scientific Python

Further information can be found at http://docs.scipy.org/doc/ (for numpy together with scipy) and http://matplotlib.org/contents.html (for plotting).

3.2.1 Creating matrices: help, array, matrix, ndim, shape, reshape, vstack, hstack, tolist, ix_

With numpy, 2 kinds of arrays are available:

matrix objects which is 2-dimensional array very similar to mathematical matrices
array objects which are more general n-dimensional arrays

Same processing could involved different methods depending on the objects used.

Here are code examples:

```
import numpy as np
help(np.array)
A=np.array([1,2,3])
B=np.matrix([[4,5,6]])
A.shape
#[Out]# (3,)
A
#[Out]# array([1, 2, 3])
B
#[Out]# matrix([[4, 5, 6]])
B.shape
#[Out]# (1, 3)
C=A+B
C
#[Out]# matrix([[5, 7, 9]])
type(C)
#[Out]# numpy.matrixlib.defmatrix.matrix
C.shape
#[Out]# (1, 3)
np.ndim(C)
#[Out]# 2
np.vstack([A,B])
#[Out]# matrix([[1, 2, 3],
#[Out]#         [4, 5, 6]])
np.hstack([np.matrix(A),B])
#[Out]# matrix([[1, 2, 3, 4, 5, 6]])
C.tolist()
#[Out]# [[5, 7, 9]]
C.tolist()[0]
#[Out]# [5, 7, 9]
np.array(C).reshape(-1,)
#[Out]# array([5, 7, 9])
A=np.array([A])
A
#[Out]# array([[1, 2, 3]])
np.hstack([A,B])
```

```
#[Out]# matrix([[1, 2, 3, 4, 5, 6]])
D=np.array([[1,2],[3,4]])
D
#[Out]# array([[1, 2],
#[Out]#        [3, 4]])
A[0,−1]
#[Out]# 3
A[0,−1]
#[Out]# 3
D[1,0]
#[Out]# 3
D[:,1]
#[Out]# array([2, 4])
D[1,:]
#[Out]# array([3, 4])
F=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]])
F
#[Out]# array([[ 1,  2,  3,  4],
#[Out]#        [ 5,  6,  7,  8],
#[Out]#        [ 9, 10, 11, 12],
#[Out]#        [13, 14, 15, 16]])
F[1::2,1:3]
#[Out]# array([[ 6,  7],
#[Out]#        [14, 15]])
np.ix_([1,2,3],[0,2])
#[Out]# (array([[1],
#[Out]#         [2],
#[Out]#         [3]]), array([[0, 2]]))
F[np.ix_([1,2,3],[0,2])]
#[Out]# array([[ 5,  7],
#[Out]#        [ 9, 11],
#[Out]#        [13, 15]])
np.r_[0:4,7:9]
#[Out]# array([0, 1, 2, 3, 7, 8])
np.r_[:4,0]
#[Out]# array([0, 1, 2, 3, 0])
F[np.r_[:4,0]]
#[Out]# array([[ 1,  2,  3,  4],
#[Out]#        [ 5,  6,  7,  8],
#[Out]#        [ 9, 10, 11, 12],
#[Out]#        [13, 14, 15, 16],
#[Out]#        [ 1,  2,  3,  4]])
F[::−1,:]
#[Out]# array([[13, 14, 15, 16],
#[Out]#        [ 9, 10, 11, 12],
#[Out]#        [ 5,  6,  7,  8],
#[Out]#        [ 1,  2,  3,  4]])
```

MATLAB	numpy.array	numpy.matrix	Notes
`ndims(a)`	`ndim(a)` or `a.ndim`		get the number of dimensions of a (tensor rank)
`size(a)`	`shape(a)` or `a.shape`		get the "size" of the matrix
`size(a,n)`	`a.shape[n-1]`		get the number of elements of the *n*th dimension of array a. (Note that MATLAB® uses 1 based indexing while Python uses 0 based indexing, *See note 'INDEXING'*)
`[ 1 2 3; 4 5 6 ]`	`array([[1.,2.,3.], [4.,5.,6.]])`	`mat([[1.,2.,3.], [4.,5.,6.]])` or `mat("1 2 3; 4 5 6")`	2x3 matrix literal
`[ a b; c d ]`	`vstack([hstack([a,b]), hstack([c,d])])`	`bmat('a b; c d')`	construct a matrix from blocks a,b,c, and d
`a(end)`	`a[-1]`	`a[:,-1][0,0]`	access last element in the 1xn matrix `a`
`a(2,5)`	`a[1,4]`		access element in second row, fifth column

Figure 3.2 Matlab equivalence

MATLAB	numpy.array	numpy.matrix	Notes
`a(2,:)`	`a[1]` or `a[1,:]`		entire second row of `a`
`a(1:5,:)`	`a[0:5]` or `a[:5]` or `a[0:5,:]`		the first five rows of `a`
`a(end-4:end,:)`	`a[-5:]`		the last five rows of `a`
`a(1:3,5:9)`	`a[0:3][:,4:9]`		rows one to three and columns five to nine of `a`. This gives read-only access.
`a([2,4,5],[1,3])`	`a[ix_([1,3,4],[0,2])]`		rows 2,4 and 5 and columns 1 and 3. This allows the matrix to be modified, and doesn't require a regular slice.
`a(3:2:21,:)`	`a[ 2:21:2,:]`		every other row of `a`, starting with the third and going to the twenty-first

Figure 3.3 Matlab equivalence

MATLAB	numpy.array	numpy.matrix	Notes
`a(1:2:end,:)`	`a[ ::2,:]`		every other row of `a`, starting with the first
`a(end:-1:1,:)` or `flipud(a)`	`a[ ::-1,:]`		`a` with rows in reverse order
`a([1:end 1],:)`	`a[r_[:len(a),0]]`		`a` with copy of the first row appended to the end

Figure 3.4 Matlab equivalence

3.2.2 Transforming matrices: dot, multiply, *, power, **, /, >, nonzero, copy, flatten

Here are code examples:

```
import numpy as np
F=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]])
F
#[Out]# array([[ 1,  2,  3,  4],
#[Out]#        [ 5,  6,  7,  8],
#[Out]#        [ 9, 10, 11, 12],
#[Out]#        [13, 14, 15, 16]])
G=np.array([[1,4,7],[3,1,2],[9,5,6],[8,4,1]])
G
#[Out]# array([[1, 4, 7],
#[Out]#        [3, 1, 2],
#[Out]#        [9, 5, 6],
#[Out]#        [8, 4, 1]])
G.transpose()
#[Out]# array([[1, 3, 9, 8],
#[Out]#        [4, 1, 5, 4],
#[Out]#        [7, 2, 6, 1]])
G
#[Out]# array([[1, 4, 7],
#[Out]#        [3, 1, 2],
#[Out]#        [9, 5, 6],
#[Out]#        [8, 4, 1]])
np.dot(F,G)
#[Out]# array([[ 66,  37,  33],
#[Out]#        [150,  93,  97],
#[Out]#        [234, 149, 161],
#[Out]#        [318, 205, 225]])
F*G  # error
F*F
#[Out]# array([[  1,   4,   9,  16],
#[Out]#        [ 25,  36,  49,  64],
#[Out]#        [ 81, 100, 121, 144],
#[Out]#        [169, 196, 225, 256]])
F=np.matrix(F)
F
#[Out]# matrix([[ 1,  2,  3,  4],
```

```
#[Out]#        [ 5,  6,  7,  8],
#[Out]#        [ 9, 10, 11, 12],
#[Out]#        [13, 14, 15, 16]])
G=np.matrix(G)
G
#[Out]# matrix([[1, 4, 7],
#[Out]#        [3, 1, 2],
#[Out]#        [9, 5, 6],
#[Out]#        [8, 4, 1]])
F*G
#[Out]# matrix([[ 66,  37,  33],
#[Out]#        [150,  93,  97],
#[Out]#        [234, 149, 161],
#[Out]#        [318, 205, 225]])
F*F
#[Out]# matrix([[ 90, 100, 110, 120],
#[Out]#        [202, 228, 254, 280],
#[Out]#        [314, 356, 398, 440],
#[Out]#        [426, 484, 542, 600]])
np.multiply(F,F)
#[Out]# matrix([[  1,   4,   9,  16],
#[Out]#        [ 25,  36,  49,  64],
#[Out]#        [ 81, 100, 121, 144],
#[Out]#        [169, 196, 225, 256]])
G=F[:,::-1]
G
#[Out]# matrix([[ 4,  3,  2,  1],
#[Out]#        [ 8,  7,  6,  5],
#[Out]#        [12, 11, 10,  9],
#[Out]#        [16, 15, 14, 13]])
X=G/F # compute X / F*X=G
X
#[Out]# matrix([[ 4.        ,  1.5       ,  0.66666667,  0.25      ],
#[Out]#        [ 1.6       ,  1.16666667,  0.85714286,  0.625     ],
#[Out]#        [ 1.33333333,  1.1       ,  0.90909091,  0.75      ],
#[Out]#        [ 1.23076923,  1.07142857,  0.93333333,  0.8125    ]])
F**2
#[Out]# matrix([[ 90, 100, 110, 120],
#[Out]#        [202, 228, 254, 280],
#[Out]#        [314, 356, 398, 440],
#[Out]#        [426, 484, 542, 600]])
F**3
#[Out]# matrix([[ 3140,  3560,  3980,  4400],
#[Out]#        [ 7268,  8232,  9196, 10160],
#[Out]#        [11396, 12904, 14412, 15920],
#[Out]#        [15524, 17576, 19628, 21680]])
np.power(F,3)
#[Out]# matrix([[   1,    8,   27,   64],
#[Out]#        [ 125,  216,  343,  512],
#[Out]#        [ 729, 1000, 1331, 1728],
#[Out]#        [2197, 2744, 3375, 4096]])
F>4
#[Out]# matrix([[False, False, False, False],
#[Out]#        [ True,  True,  True,  True],
#[Out]#        [ True,  True,  True,  True],
```

```
#[Out]#       [ True,  True,  True,  True]], dtype=bool)
np.nonzero(F>4)
#[Out]# (array([1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3]),
#[Out]#  array([0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]))
F[F>4]
#[Out]# matrix([[ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16]])
A[:]=3
E=np.ones((2,3))
E
#[Out]# array([[ 1.,  1.,  1.],
#[Out]#       [ 1.,  1.,  1.]])
E[:]=3
E
#[Out]# array([[ 3.,  3.,  3.],
#[Out]#       [ 3.,  3.,  3.]])
H=F
F
#[Out]# matrix([[ 1,  2,  3,  4],
#[Out]#       [ 5,  6,  7,  8],
#[Out]#       [ 9, 10, 11, 12],
#[Out]#       [13, 14, 15, 16]])
H[1,1]=-1
F
#[Out]# matrix([[ 1,  2,  3,  4],
#[Out]#       [ 5, -1,  7,  8],
#[Out]#       [ 9, 10, 11, 12],
#[Out]#       [13, 14, 15, 16]])
H=F.copy()
H
#[Out]# matrix([[ 1,  2,  3,  4],
#[Out]#       [ 5, -1,  7,  8],
#[Out]#       [ 9, 10, 11, 12],
#[Out]#       [13, 14, 15, 16]])
H[1,1]=-2
F
#[Out]# matrix([[ 1,  2,  3,  4],
#[Out]#       [ 5, -1,  7,  8],
#[Out]#       [ 9, 10, 11, 12],
#[Out]#       [13, 14, 15, 16]])
I=F[1:3,:]
I[:]=3
I=F[1:3,:]
I
#[Out]# matrix([[3, 3, 3, 3],
#[Out]#       [3, 3, 3, 3]])
F
#[Out]# matrix([[ 1,  2,  3,  4],
#[Out]#       [ 3,  3,  3,  3],
#[Out]#       [ 3,  3,  3,  3],
#[Out]#       [13, 14, 15, 16]])
I=F[:,1:3].copy()
I
#[Out]# matrix([[ 2,  3],
#[Out]#       [ 3,  3],
#[Out]#       [ 3,  3],
```

```
#[Out]#        [14, 15]])
I[:]=2
F
#[Out]# matrix([[ 1,  2,  3,  4],
#[Out]#         [ 3,  3,  3,  3],
#[Out]#         [ 3,  3,  3,  3],
#[Out]#         [13, 14, 15, 16]])
F.flatten()
#[Out]# matrix([[ 1,  2,  3,  4,  3,  3,  3,  3,  3,  3,  3,  3, 13, 14, 15, 16]])
```

MATLAB	numpy.array	numpy.matrix	Notes
`a.'`	`a.transpose()` or `a.T`		transpose of `a`
`a'`	`a.conj().transpose()` or `a.conj().T`	`a.H`	conjugate transpose of `a`
`a * b`	`dot(a,b)`	`a * b`	matrix multiply
`a .* b`	`a * b`	`multiply(a,b)`	element-wise multiply
`a./b`	`a/b`		element-wise divide
`a.^3`	`a**3`	`power(a,3)`	element-wise exponentiation

Figure 3.5 Matlab equivalence

MATLAB	numpy.array	numpy.matrix	Notes
`(a>0.5)`	`(a>0.5)`		matrix whose i,jth element is (a_ij > 0.5)
`find(a>0.5)`	`nonzero(a>0.5)`		find the indices where (a > 0.5)
`a(:,find(v>0.5))`	`a[:,nonzero(v>0.5)[0]]`	`a[:,nonzero(v.A>0.5)[0]]`	extract the columms of a where vector v > 0.5
`a(:,find(v>0.5))`	`a[:,v.T>0.5]`	`a[:,v.T>0.5)]`	extract the columms of a where column vector v > 0.5
`a(a<0.5)=0`	`a[a<0.5]=0`		a with elements less than 0.5 zeroed out
`a .* (a>0.5)`	`a * (a>0.5)`	`mat(a.A * (a>0.5).A)`	a with elements less than 0.5 zeroed out
`a(:) = 3`	`a[:] = 3`		set all values to the same scalar value
`y=x`	`y = x.copy()`		numpy assigns by reference
`y=x(2,:)`	`y = x[1,:].copy()`		numpy slices are by reference
`y=x(:)`	`y = x.flatten(1)`		turn array into vector (note that this forces a copy)

Figure 3.6 Matlab equivalence

3.2.3 Generating matrices: r_, zeros, ones, eye, diag, rand, randn, randint, max, maximum, norm, logical_and, logical_or, &, |

Here are some code examples:

```
import numpy as np
np.r_[:11]
#[Out]# array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
np.r_['r',:11]
#[Out]# matrix([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]])
np.r_['c',:11]
#[Out]# matrix([[ 0],
#[Out]#         [ 1],
#[Out]#         [ 2],
#[Out]#         [ 3],
#[Out]#         [ 4],
#[Out]#         [ 5],
#[Out]#         [ 6],
#[Out]#         [ 7],
#[Out]#         [ 8],
#[Out]#         [ 9],
#[Out]#         [10]])
np.zeros((3,4))
#[Out]# array([[ 0.,  0.,  0.,  0.],
#[Out]#        [ 0.,  0.,  0.,  0.],
#[Out]#        [ 0.,  0.,  0.,  0.]])
np.ones((3,4))
#[Out]# array([[ 1.,  1.,  1.,  1.],
#[Out]#        [ 1.,  1.,  1.,  1.],
#[Out]#        [ 1.,  1.,  1.,  1.]])
np.eye(3)
#[Out]# array([[ 1.,  0.,  0.],
#[Out]#        [ 0.,  1.,  0.],
#[Out]#        [ 0.,  0.,  1.]])
np.diag([3,4,5])
#[Out]# array([[3, 0, 0],
#[Out]#        [0, 4, 0],
#[Out]#        [0, 0, 5]])
np.random.rand(3,2) #uniform distribution in [0,1]
#[Out]# array([[ 0.84549559,  0.63542167],
#[Out]#        [ 0.82286242,  0.62421155],
#[Out]#        [ 0.77991685,  0.54877061]])
np.random.randn(3,2) #gaussian mu=0 sigma=1
#[Out]# array([[ 0.33571829, -0.6069386 ],
#[Out]#        [-0.87850092, -0.17938046],
#[Out]#        [ 1.00472881,  1.38929716]])
np.random.randint(2,9,(3,2)) # uniform
#[Out]# array([[7, 7],
#[Out]#        [7, 8],
#[Out]#        [8, 7]])
F=np.random.randint(2,9,(3,2)) # uniform
F
#[Out]# array([[7, 7],
#[Out]#        [2, 3],
```

```
#[Out]#       [4, 6]])
F.max(0)
#[Out]# array([7, 7])
F.max(1)
#[Out]# array([7, 3, 6])
F.max()
#[Out]# 7
G=np.random.randint(2,9,(3,2)) # uniform
G
#[Out]# array([[6, 8],
#[Out]#       [2, 5],
#[Out]#       [7, 3]])
np.maximum(F,G)
#[Out]# array([[7, 8],
#[Out]#       [2, 5],
#[Out]#       [7, 6]])
np.linalg.norm(F)
#[Out]# 12.767145334803704
H=np.random.randint(0,2,(2,3))
H
#[Out]# array([[1, 0, 0],
#[Out]#       [0, 0, 0]])
I=np.random.randint(0,2,(2,3))
I
#[Out]# array([[1, 1, 1],
#[Out]#       [1, 1, 0]])
np.logical_and(H,I)
#[Out]# array([[ True, False, False],
#[Out]#       [False, False, False]], dtype=bool)
np.logical_or(H,I)
#[Out]# array([[ True,  True,  True],
#[Out]#       [ True,  True, False]], dtype=bool)
H&I
#[Out]# array([[1, 0, 0],
#[Out]#       [0, 0, 0]])
H|I
#[Out]# array([[1, 1, 1],
#[Out]#       [1, 1, 0]])
```

MATLAB	numpy.array	numpy.matrix	Notes
`1:10`	`arange(1.,11.)` or `r_[1.:11.]` or `r_[1:10:10j]`	`mat(arange(1.,11.))` or `r_[1.:11.,'r']`	create an increasing vector *see note 'RANGES'*
`0:9`	`arange(10.)` or `r_[:10.]` or `r_[:9:10j]`	`mat(arange(10.))` or `r_[:10.,'r']`	create an increasing vector *see note 'RANGES'*
`[1:10]'`	`arange(1.,11.)[:, newaxis]`	`r_[1.:11.,'c']`	create a column vector
`zeros(3,4)`	`zeros((3,4))`	`mat(...)`	3x4 rank-2 array full of 64-bit floating point zeros
`zeros(3,4,5)`	`zeros((3,4,5))`	`mat(...)`	3x4x5 rank-3 array full of 64-bit floating point zeros
`ones(3,4)`	`ones((3,4))`	`mat(...)`	3x4 rank-2 array full of 64-bit floating point ones
`eye(3)`	`eye(3)`	`mat(...)`	3x3 identity matrix
`diag(a)`	`diag(a)`	`mat(...)`	vector of diagonal elements of a
`diag(a,0)`	`diag(a,0)`	`mat(...)`	square diagonal matrix whose nonzero values are the elements of a
`rand(3,4)`	`random.rand(3,4)`	`mat(...)`	random 3x4 matrix

Figure 3.7 Matlab equivalence

MATLAB	numpy.array	numpy.matrix	Notes
`max(a)`	`a.max(0)`		maximum element of each column of matrix a
`max(a,[],2)`	`a.max(1)`		maximum element of each row of matrix a
`max(a,b)`	`maximum(a, b)`		compares a and b element-wise, and returns the maximum value from each pair
`norm(v)`	`sqrt(dot(v,v))` or `Sci.linalg.norm(v)` or `linalg.norm(v)`	`sqrt(dot(v.A,v.A))` or `Sci.linalg.norm(v)` or `linalg.norm(v)`	L2 norm of vector v
`a & b`	`logical_and(a,b)`		element-by-element AND operator (Numpy ufunc) *see note 'LOGICOPS'*
`a | b`	`logical_or(a,b)`		element-by-element OR operator (Numpy ufunc) *see note 'LOGICOPS'*

Figure 3.8 Matlab equivalence

MATLAB	numpy.array	numpy.matrix	Notes
`bitand(a,b)`	`a & b`		bitwise AND operator (Python native and Numpy ufunc)
`bitor(a,b)`	`a | b`		bitwise OR operator (Python native and Numpy ufunc)

Figure 3.9 Matlab equivalence

3.2.4 Calculating with matrices: inv, pinv, matrix_rank, solve, lstsq, svd, transpose, eig, sort, linspace, meshgrid, mgrid, ogrid, concatenate, tile, squeeze, integrate

Here are code examples:

```
import numpy as np
import numpy.linalg as linalg
D=np.array([[1,2],[3,4]])
linalg.inv(D)
#[Out]# array([[−2. ,  1. ],
#[Out]#        [ 1.5, −0.5]])
np.dot(D, linalg.inv(D))
#[Out]# array([[  1.00000000e+00,   0.00000000e+00],
#[Out]#        [  8.88178420e−16,   1.00000000e+00]])
I=np.random.randint(0,2,(2,3))
I
#[Out]# array([[0, 0, 0],
#[Out]#        [1, 0, 0]])
linalg.pinv(I)
#[Out]# array([[ 0.,  1.],
#[Out]#        [ 0.,  0.],
#[Out]#        [ 0.,  0.]])
np.dot(I,linalg.pinv(I))
#[Out]# array([[ 0.,  0.],
#[Out]#        [ 0.,  1.]])
linalg.matrix_rank(I)
#[Out]# 1
H=np.random.randint(0,2,(2,3))
H
#[Out]# array([[0, 0, 0],
#[Out]#        [0, 1, 0]])
linalg.matrix_rank(H)
#[Out]# 1
A=np.array([[1,2],[−2,1]])
B=np.array([[3],[−1]])
linalg.solve(A,B) #solve x1+2x2=3,−2x1+x2=−1
#[Out]# array([[ 1.],
#[Out]#        [ 1.]])
#x1=1, x2=1
A=np.random.randint(0,10,(3,2))
```

```
A
#[Out]# array([[9, 0],
#[Out]#        [4, 1],
#[Out]#        [6, 9]])
B=np.random.randint(0,10,(3,1))
B
#[Out]# array([[8],
#[Out]#        [5],
#[Out]#        [8]])
linalg.lstsq(A,B) #solve Ax~B (least square)
#[Out]# (array([[ 0.92999204],
#[Out]#         [ 0.28122514]]),
#[Out]#  array([ 1.14677804]),
#[Out]#  2,
#[Out]#  array([ 13.07127037,   6.64393639]))
# best solution is x1=0.92999204, x2=0.28122514
J=np.random.randint(0,11,(3,4))
J
#[Out]# array([[ 3,  4,  0,  7],
#[Out]#        [ 2, 10,  4,  6],
#[Out]#        [ 9,  4,  0,  8]])
(U,S,V)=linalg.svd(J)
U
#[Out]# array([[−0.45603171,  0.09977533,  0.88435285],
#[Out]#        [−0.61454651, −0.75404815, −0.23182748],
#[Out]#        [−0.64371396,  0.64919664, −0.40518645]])
S
#[Out]# array([ 18.22998676,   7.28665454,   2.36056104])
V
#[Out]# array([[−0.4602644 , −0.57841226, −0.134843  , −0.65985856],
#[Out]#        [ 0.63595706, −0.62368726, −0.4139338 ,  0.18770089],
#[Out]#        [−0.61734242, −0.17013296, −0.39283455,  0.66001828],
#[Out]#        [−0.05102585, −0.497502  ,  0.8100353 ,  0.30615508]])
np.dot(U.transpose(),U)
#[Out]# array([[  1.00000000e+00,   5.55111512e−17,   0.00000000e+00],
#[Out]#        [  5.55111512e−17,   1.00000000e+00,   0.00000000e+00],
#[Out]#        [  0.00000000e+00,   0.00000000e+00,   1.00000000e+00]])
np.dot(V.transpose(),V)
#[Out]# array([[  1.00000000e+00,   4.44089210e−16,   1.38777878e−16,
#[Out]#           0.00000000e+00],
#[Out]#        [  4.44089210e−16,   1.00000000e+00,   1.11022302e−16,
#[Out]#          −1.11022302e−16],
#[Out]#        [  1.38777878e−16,   1.11022302e−16,   1.00000000e+00,
#[Out]#          −9.71445147e−17],
#[Out]#        [  0.00000000e+00,  −1.11022302e−16,  −9.71445147e−17,
#[Out]#           1.00000000e+00]])
D=np.hstack([np.diag(S),np.zeros((3,1))])
D
#[Out]# array([[ 18.22998676,   0.        ,   0.        ,   0.        ],
#[Out]#        [  0.        ,   7.28665454,   0.        ,   0.        ],
#[Out]#        [  0.        ,   0.        ,   2.36056104,   0.        ]])
np.dot(np.dot(U,D),V)
#[Out]# array([[  3.00000000e+00,   4.00000000e+00,  −9.99200722e−16,
#[Out]#           7.00000000e+00],
#[Out]#        [  2.00000000e+00,   1.00000000e+01,   4.00000000e+00,
```

```
#[Out]#          6.00000000e+00],
#[Out]#       [ 9.00000000e+00,  4.00000000e+00,  1.60982339e-15,
#[Out]#          8.00000000e+00]])
L=np.matrix([[6,-3,-10],[0,2,-2],[2,-1,-5]])
L
#[Out]# matrix([[ 6, -3, -10],
#[Out]#         [ 0,  2,  -2],
#[Out]#         [ 2, -1,  -5]])
V,P=linalg.eig(L)
V
#[Out]# array([-2.88089912,  4.24581024,  1.63508888])
P
#[Out]# matrix([[ 0.76017728,  0.95227747, -0.73605421],
#[Out]#         [ 0.24634855, -0.20299732, -0.66592894],
#[Out]#         [ 0.60120121,  0.22794673, -0.12150244]])
P*np.diag(V)*linalg.inv(P)
#[Out]# matrix([[ 6.00000000e+00, -3.00000000e+00, -1.00000000e+01],
#[Out]#         [-1.33226763e-15,  2.00000000e+00, -2.00000000e+00],
#[Out]#         [ 2.00000000e+00, -1.00000000e+00, -5.00000000e+00]])
L0=L.copy()
L1=L.copy()
L0.sort(0)
L0
#[Out]# matrix([[ 0, -3, -10],
#[Out]#         [ 2, -1,  -5],
#[Out]#         [ 6,  2,  -2]])
L1.sort(1)
L1
#[Out]# matrix([[-10, -3,  6],
#[Out]#         [ -2,  0,  2],
#[Out]#         [ -5, -1,  2]])
np.linspace(1,3,4)
#[Out]# array([ 1.        ,  1.66666667,  2.33333333,  3.        ])
np.meshgrid([1,2,4],[2,4,5])
#[Out]# [array([[1, 2, 4],
#[Out]#         [1, 2, 4],
#[Out]#         [1, 2, 4]]), array([[2, 2, 2],
#[Out]#         [4, 4, 4],
#[Out]#         [5, 5, 5]])]
np.mgrid[0:9,0:6]
#[Out]# array([[[0, 0, 0, 0, 0, 0],
#[Out]#         [1, 1, 1, 1, 1, 1],
#[Out]#         [2, 2, 2, 2, 2, 2],
#[Out]#         [3, 3, 3, 3, 3, 3],
#[Out]#         [4, 4, 4, 4, 4, 4],
#[Out]#         [5, 5, 5, 5, 5, 5],
#[Out]#         [6, 6, 6, 6, 6, 6],
#[Out]#         [7, 7, 7, 7, 7, 7],
#[Out]#         [8, 8, 8, 8, 8, 8]],
#[Out]#
#[Out]#        [[0, 1, 2, 3, 4, 5],
#[Out]#         [0, 1, 2, 3, 4, 5],
#[Out]#         [0, 1, 2, 3, 4, 5],
#[Out]#         [0, 1, 2, 3, 4, 5],
#[Out]#         [0, 1, 2, 3, 4, 5],
```

```
#[Out]#       [0, 1, 2, 3, 4, 5],
#[Out]#       [0, 1, 2, 3, 4, 5],
#[Out]#       [0, 1, 2, 3, 4, 5],
#[Out]#       [0, 1, 2, 3, 4, 5]])
np.ogrid[0:3,0:4]
#[Out]# [array([[0],
#[Out]#       [1],
#[Out]#       [2]]), array([[0, 1, 2, 3]])]
np.concatenate((L0,L1),0)
#[Out]# matrix([[ 0, −3, −10],
#[Out]#       [ 2, −1, −5],
#[Out]#       [ 6,  2, −2],
#[Out]#       [−10, −3,  6],
#[Out]#       [ −2,  0,  2],
#[Out]#       [ −5, −1,  2]])
np.concatenate((L0,L1),1)
#[Out]# matrix([[ 0, −3, −10, −10, −3,  6],
#[Out]#       [ 2, −1, −5, −2,  0,  2],
#[Out]#       [ 6,  2, −2, −5, −1,  2]])
np.tile(L0,(2,3))
#[Out]# matrix([[ 0, −3, −10,  0, −3, −10,  0, −3, −10],
#[Out]#       [ 2, −1, −5,  2, −1, −5,  2, −1, −5],
#[Out]#       [ 6,  2, −2,  6,  2, −2,  6,  2, −2],
#[Out]#       [ 0, −3, −10,  0, −3, −10,  0, −3, −10],
#[Out]#       [ 2, −1, −5,  2, −1, −5,  2, −1, −5],
#[Out]#       [ 6,  2, −2,  6,  2, −2,  6,  2, −2]])
F=np.array([[1,2,3]])
F
#[Out]# array([[1, 2, 3]])
F.squeeze()
#[Out]# array([1, 2, 3])
F=np.array([[1],[2],[3]])
F
#[Out]# array([[1],
#[Out]#       [2],
#[Out]#       [3]])
F.squeeze()
#[Out]# array([1, 2, 3])
```

MATLAB	numpy.array	numpy.matrix	Notes
`inv(a)`	`linalg.inv(a)`		inverse of square matrix a
`pinv(a)`	`linalg.pinv(a)`		pseudo-inverse of matrix a
`rank(a)`	`linalg.matrix_rank(a)`		rank of a matrix a
`a\b`	`linalg.solve(a,b)` if a is square `linalg.lstsq(a,b)` otherwise		solution of a x = b for x
`b/a`	Solve a.T x.T = b.T instead		solution of x a = b for x
`[U,S,V]=svd(a)`	`U, S, Vh = linalg.svd(a), V = Vh.T`		singular value decomposition of a

Figure 3.10 Matlab equivalence

MATLAB	numpy.array	numpy.matrix	Notes
`chol(a)`	`linalg.cholesky(a).T`		cholesky factorization of a matrix (chol(a) in matlab returns an upper triangular matrix, but linalg.cholesky(a) returns a lower triangular matrix)
`[V,D]=eig(a)`	`D,V = linalg.eig(a)`		eigenvalues and eigenvectors of a
`[V,D]=eig(a,b)`	`V,D = Sci.linalg.eig(a,b)`		eigenvalues and eigenvectors of a,b
`[V,D]=eigs(a,k)`			find the k largest eigenvalues and eigenvectors of a
`[Q,R,P]=qr(a,0)`	`Q,R = Sci.linalg.qr(a)`	`mat(...)`	QR decomposition
`[L,U,P]=lu(a)`	`L,U = Sci.linalg.lu(a)` or `LU,P=Sci.linalg.lu_factor(a)`	`mat(...)`	LU decomposition (note: P(Matlab) == transpose(P(numpy)))
`conjgrad`	`Sci.linalg.cg`	`mat(...)`	Conjugate gradients solver

Figure 3.11 Matlab equivalence

MATLAB	numpy.array	numpy.matrix	Notes
`fft(a)`	`fft(a)`	`mat(...)`	Fourier transform of a
`ifft(a)`	`ifft(a)`	`mat(...)`	inverse Fourier transform of a
`sort(a)`	`sort(a)` or `a.sort()`	`mat(...)`	sort the matrix
`[b,I] = sortrows(a,i)`	`I = argsort(a[:,i]), b=a[I,:]`		sort the rows of the matrix
`regress(y,X)`	`linalg.lstsq(X,y)`		multilinear regression
`decimate(x, q)`	`Sci.signal.resample(x, len(x)/q)`		downsample with low-pass filtering
`unique(a)`	`unique(a)`		
`squeeze(a)`	`a.squeeze()`		

Figure 3.12 Matlab equivalence

MATLAB	numpy.array	numpy.matrix	Notes
`linspace(1,3,4)`	`linspace(1,3,4)`	`mat(...)`	4 equally spaced samples between 1 and 3, inclusive
`[x,y]=meshgrid(0:8,0:5)`	`mgrid[0:9.,0:6.]` or `meshgrid(r_[0:9.],r_[0:6.]`	`mat(...)`	two 2D arrays: one of x values, the other of y values
	`ogrid[0:9.,0:6.]` or `ix_(r_[0:9.],r_[0:6.]`	`mat(...)`	the best way to eval functions on a grid
`[x,y]=meshgrid([1,2,4], [2,4,5])`	`meshgrid([1,2,4],[2,4,5])`	`mat(...)`	
	`ix_([1,2,4],[2,4,5])`	`mat(...)`	the best way to eval functions on a grid
`repmat(a, m, n)`	`tile(a, (m, n))`	`mat(...)`	create m by n copies of a
`[a b]`	`concatenate((a,b),1)` or `hstack((a,b))` or `column_stack((a,b))` or `c_[a,b]`	`concatenate((a,b),1)`	concatenate columns of a and b
`[a; b]`	`concatenate((a,b))` or `vstack((a,b))` or `r_[a,b]`	`concatenate((a,b))`	concatenate rows of a and b

Figure 3.13 Matlab equivalence

3.3 Scientific Python for advance computions with scipy

'scipy' module is a natural extension of numpy. It contains more advanced functions

3.3.1 Load and saving matrices: save, load, savetxt, loadtxt, savemat, loadmat, expm, interp1d

```
# use numpy own format for one matrix
import numpy as np
import scipy.io
data=np.random.randn(2,1000)
np.save('pop.npy', data)
data1 = np.load('pop.npy')
# use text format for one matrix
np.savetxt('pop2.txt', data)
data2 = np.loadtxt('pop2.txt')
# use matlab format for several matrices ({'var1':matrix1,'var2',...})
scipy.io.savemat('pop3.mat',{'data':data})
data3=scipy.io.loadmat('pop3.mat')
```

3.3.2 Some advanced computations

Computing matrix exponential can be managed in this way:

```
 from pylab import *
import scipy.linalg
A=matrix([[1,2],[3,4]])
scipy.linalg.expm(A)
#[Out]# array([[  51.9689562 ,   74.73656457],
#[Out]#        [ 112.10484685,  164.07380305]])
```

Interpolating is made easy:

```
 from scipy import interpolate
import matplotlib.pyplot as plt
x = np.arange(0, 10)
y = np.exp(-x/3.)
f = interpolate.interp1d(x, y)
xnew = np.arange(0,9, 0.1)
ynew = f(xnew)
plt.plot(x, y, 'o', xnew, ynew, '-')
plt.show()
```

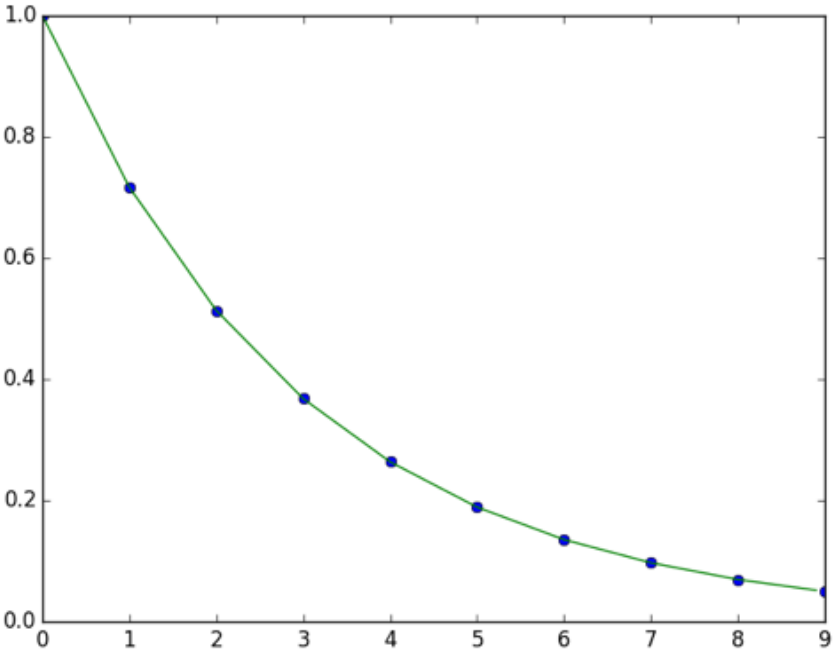

Figure 3.14 Interpolating

In the module 'scipy.constants', 'scipy' embeds physical constants (see http://docs.scipy.org/doc/scipy-0.14.0/reference/constants.html) and some conversion functions, for example:

C2K(C) Convert Celsius to Kelvin
K2C(K) Convert Kelvin to Celsius
F2C(F) Convert Fahrenheit to Celsius
C2F(C) Convert Celsius to Fahrenheit
F2K(F) Convert Fahrenheit to Kelvin
K2F(K) Convert Kelvin to Fahrenheit

Exercise 3.1 Matrix computations

Enter the following matrices:

$$u_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, u_2 = \begin{bmatrix} -5 \\ 2 \\ 1 \end{bmatrix}, u_3 = \begin{bmatrix} -1 \\ -3 \\ 7 \end{bmatrix}, A = \begin{bmatrix} 2 & 3 & 4 \\ 7 & 6 & 5 \\ 2 & 8 & 7 \end{bmatrix}$$

Compute $u_1 + 3u_2 - 5\frac{u_3}{5}$
Compute the scalar product of u_1 and u_2
Compute $(A^2 - I_2)e^A u_1 - A^{-1}u_2$
Extract the size of A
What is the rank of A?

Solve $Ax = u_1$

Solution:

```
 from pylab import *
from scipy.linalg import expm

u1=matrix([[1],[2],[3]])
u2=matrix([[-5],[2],[1]])
u3=matrix([[-1],[-3],[7]])
A=matrix([[2,3,4],[7,6,5],[2,8,7]])

print(u1+3*u2-u3/5)
print(u1.transpose()*u2)
print((A**2-eye(3))*expm(A)*u1-inv(A)*u2)
print(A.shape)
print(matrix_rank(A))
print(solve(A,u1))
print(A*solve(A,u1))
```

Exercise 3.2 Recurrent system

Consider the following system:
Compute X_k for $k \in \{1, \ldots, 10\}$
Compute $lim_{k\to\infty} X_k$. Use eigenvalues to prove the result.

Solution:

```
from pylab import *
from numpy.linalg import *

X0=matrix([[3],[1]])
X=X0.copy()
A=matrix([[-.5,.8],[.3,.8]])
for i in range(10):
   print(X)
   X=A*X
X=X0.copy()
for i in range(1000):
   X=A*X
print('X='+str(X))
V,P=eig(A)
print(V)
print('X='+str(P*matrix_power(diag(V),1000)*inv(P)*X0))
```

$$A = P\Delta P^{-1}$$
$$A^n = (P\Delta P^{-1}) \times (P\Delta P^{-1}) \times \cdots \times (P\Delta P^{-1}) = (P\Delta^n P^{-1})$$

Exercise 3.3 Multi-dimensional solution space
Create a matrix representation of:

$$\begin{cases} 2x_1 + 3x_2 + 7x_3 + 3x_4 = 1 \\ x_1 - 2x_2 + 4x_3 + 7x_4 = 3 \\ x_2 - 5x_3 + x_4 = -1 \end{cases}$$

Prove that $\forall A,\ Ax = B \rightarrow x = (A^+A - I)\chi + A^+B; \forall \chi$ and compute the matrices of the solution.
In order to estimate the dimension of the solution space, compute the rank of $A^+A - I$.
Compute randomly some solutions and check whether $Ax = B$ is satisfied.

Solution:

```
from pylab import *
from numpy.linalg import pinv
A=matrix([[2,3,7,3],[1,-2,4,7],[0,1,-5,1]])
B=matrix([[1],[3],[-1]])
Aplus = pinv(A)
Aplus
#[Out]# matrix([[ 0.05819261, -0.01104518,  0.05636696],
#[Out]#         [ 0.18051118, -0.10798722,  0.17092652],
#[Out]#         [ 0.04016431,  0.00100411, -0.14148791],
#[Out]#         [ 0.02031036,  0.11300776,  0.12163396]])
M=(Aplus*A-eye(4))
M
#[Out]# matrix([[-0.89465997,  0.25303514,  0.08133272,  0.15362848],
#[Out]#         [ 0.25303514, -0.0715655 , -0.02300319, -0.04345048],
#[Out]#         [ 0.08133272, -0.02300319, -0.00739388, -0.01396623],
#[Out]#         [ 0.15362848, -0.04345048, -0.01396623, -0.02638065]])
X0=Aplus*B
X0
#[Out]# matrix([[-0.0313099 ],
#[Out]#         [-0.314377  ],
#[Out]#         [ 0.18466454],
#[Out]#         [ 0.23769968]])
solutions=M*randint(-10,10,(4,10))+X0
A*solutions
#[Out]# matrix([[ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.],
#[Out]#         [ 3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.],
#[Out]#         [-1., -1., -1., -1., -1., -1., -1., -1., -1., -1.]])
```

3.4 Scientific Python for plotting with matplotlib

matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and ipython shell (ala MATLAB or Mathematica), web application servers, and six graphical user interface toolkits.

matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc, with just a few lines of code.

For simple plotting the pyplot interface provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

Simple examples are provided in the next as an illustration of the matplotlib capabilities. Further documentation can be found at http://matplotlib.org/contents.html.

3.4.1 3 simple plots

This simple code:

```
import numpy
import matplotlib.pyplot as plt
x = numpy.linspace(0, 3, 20)
y = numpy.linspace(0, 9, 20)
plt.plot(x, y)
plt.plot(x, y, 'o')
plt.grid()
plt.figure()
image = numpy.random.rand(30, 30)
plt.imshow(image)
plt.gray()
plt.figure()
plt.pcolor(image)
plt.hot()
plt.colorbar()
plt.show()
```

leads to these 3 following figures:

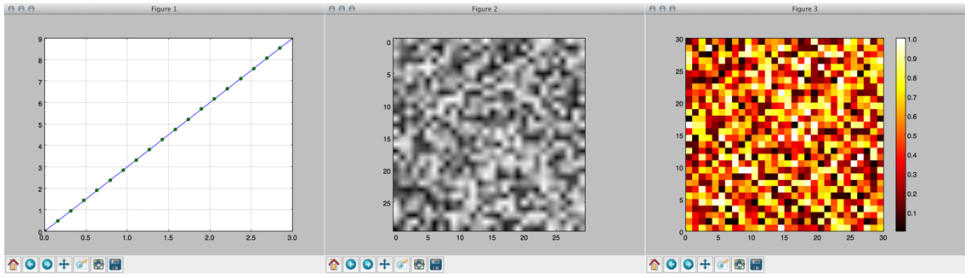

Figure 3.15 3 simple plots

3.4.2 Standard time plot

This code:

```
import numpy
from numpy.random import randn
import matplotlib.pyplot as plt
npoints=100
x=[i for i in range(npoints)]
y1 = randn(npoints)
y2 = randn(npoints)
y3 = randn(npoints)
y4 = randn(npoints)
plt.subplot(2,1,1)
plt.plot(x,y1)
plt.plot(x,y2,':')
plt.grid()
plt.xlabel('time')
plt.ylabel('series 1 and 2')
plt.axis('tight')
plt.subplot(2,1,2)
plt.plot(x,y3)
plt.plot(x,y4,'r')
plt.grid()
plt.xlabel('time')
plt.ylabel('series 3 and 4')
plt.axis('tight')
plt.show()
```

leads to:

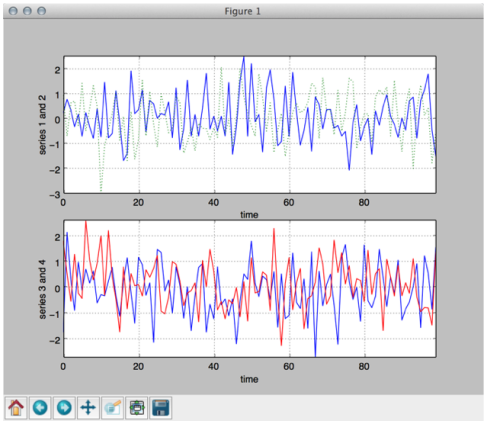

Figure 3.16 Standard time plot

3.4.3 Another time plot with modified spaces

```
from pylab import *
t = arange(0.0, 2.0, 0.01)
s1 = sin(2*pi*t)
s2 = exp(-t)
s3 = s1*s2
f = figure()
subplots_adjust(hspace=0.001)
ax1 = subplot(311)
ax1.plot(t,s1)
yticks(arange(-0.9, 1.0, 0.4))
ylim(-1,1)
ax2 = subplot(312, sharex=ax1)
ax2.plot(t,s2)
yticks(arange(0.1, 1.0,  0.2))
ylim(0,1)
ax3 = subplot(313, sharex=ax1)
ax3.plot(t,s3)
yticks(arange(-0.9, 1.0, 0.4))
ylim(-1,1)
xticklabels = ax1.get_xticklabels()+ax2.get_xticklabels()
setp(xticklabels, visible=False)
show()
```

yields:

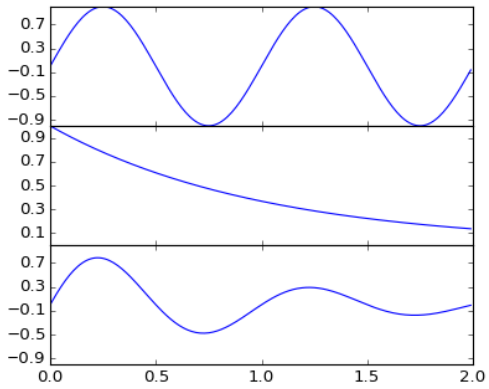

Figure 3.17 Another time plot with modified spaces

3.4.4 Plots with different strokes

```
import numpy as np
import matplotlib.pyplot as plt
a = b = np.arange(0,3, .02)
c = np.exp(a)
d = c[::-1]
plt.plot(a, c, 'k--', label='Model length')
plt.plot(a, d, 'k:', label='Data length')
plt.plot(a, c+d, 'k', label='Total message length')
```

```
legend = plt.legend(loc='upper center', shadow=True, fontsize='x-large')
legend.get_frame().set_facecolor('#00FFCC')
plt.show()
```

yields:

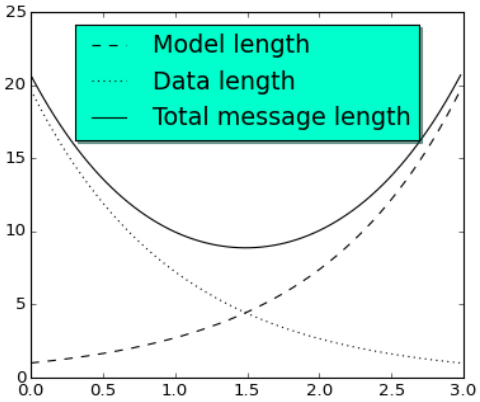

Figure 3.18 Plots with different strokes

3.4.5 Filled plot

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 1)
y = np.sin(4 * np.pi * x) * np.exp(-5 * x)
plt.fill(x, y, 'r')
plt.grid(True)
plt.show()
```

yields:

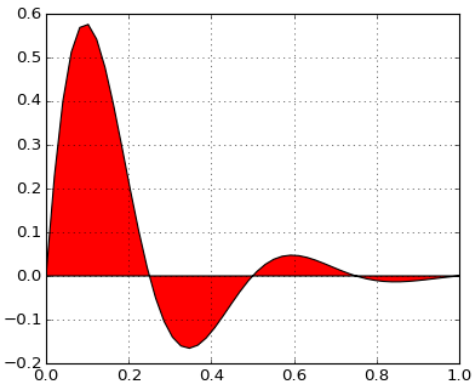

Figure 3.19 Filled plot

3.4.6 Dashed plot

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 10)
line, = plt.plot(x, np.sin(x), '--', linewidth=2)
# 10 points on, 5 off, 100 on, 5 off
dashes = [10, 5, 100, 5]
line.set_dashes(dashes)
plt.show()
```

yields:

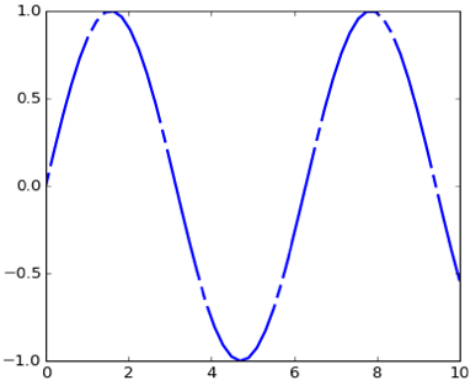

Figure 3.20 Dashed plot

3.4.7 Colored plot

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.collections as collections
```

```
t = np.arange(0.0, 2, 0.01)
s1 = np.sin(2*np.pi*t)
s2 = 1.2*np.sin(4*np.pi*t)
fig, ax = plt.subplots()
ax.set_title('using span_where')
ax.plot(t, s1, color='black')
ax.axhline(0, color='black', lw=2)
collection = collections.BrokenBarHCollection.span_where(t, ymin=0, ymax=1, where=s1>0, →
    ↪ facecolor='green', alpha=0.5)
ax.add_collection(collection)
collection = collections.BrokenBarHCollection.span_where(t, ymin=-1, ymax=0, where=s1<0, →
    ↪ facecolor='red', alpha=0.5)
ax.add_collection(collection)
plt.show()
```

yields:

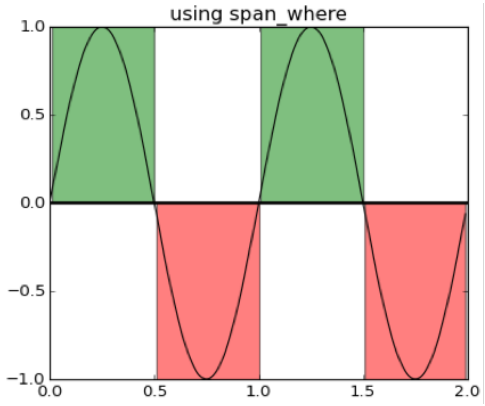

Figure 3.21 Colored plot

3.4.8 Another colored plot

```
import numpy as np
from matplotlib import pyplot as plt
fnx = lambda : np.random.randint(5, 50, 10)
y = np.row_stack((fnx(), fnx(), fnx()))
x = np.arange(10)
y1, y2, y3 = fnx(), fnx(), fnx()
fig, ax = plt.subplots()
ax.stackplot(x, y)
plt.show()
fig, ax = plt.subplots()
ax.stackplot(x, y1, y2, y3)
plt.show()
```

yields:

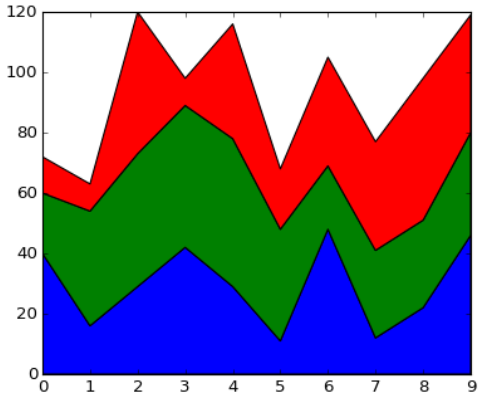

Figure 3.22 Another colored plot

3.4.9 Dotted plot

```
import numpy as np
import matplotlib.pyplot as plt
fig, ax1 = plt.subplots()
t = np.arange(0.01, 10.0, 0.01)
s1 = np.exp(t)
ax1.plot(t, s1, 'b-')
ax1.set_xlabel('time (s)')
ax1.set_ylabel('exp', color='b')
for tl in ax1.get_yticklabels():
    tl.set_color('b')
ax2 = ax1.twinx()
s2 = np.sin(2*np.pi*t)
ax2.plot(t, s2, 'r.')
ax2.set_ylabel('sin', color='r')
for tl in ax2.get_yticklabels():
    tl.set_color('r')
plt.show()
```

yields:

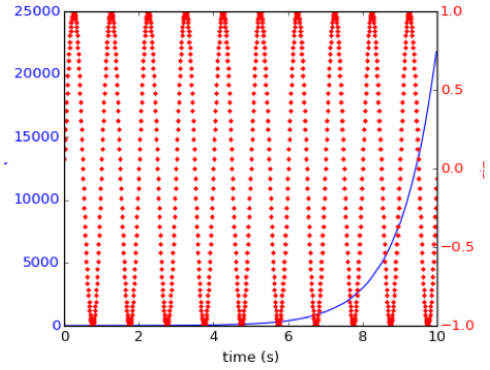

Figure 3.23 Dotted plot

3.4.10 Curve and point

```
import matplotlib.path as mpath
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
Path = mpath.Path
fig, ax = plt.subplots()
pp1 = mpatches.PathPatch(
   Path([(0, 0), (1, 0), (1, 1), (0, 0)],[Path.MOVETO, Path.CURVE3, Path.CURVE3, Path. →
      ↪ CLOSEPOLY]),fc="none", transform=ax.transData)
ax.add_patch(pp1)
ax.plot([0.75], [0.25], "ro")
ax.set_title('The red point should be on the path')
plt.show()
```

yields:

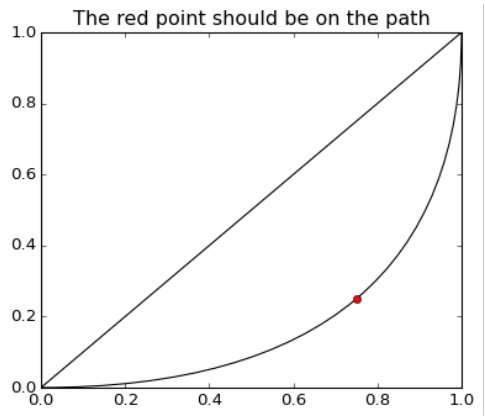

Figure 3.24 Curve and point

3.4.11 Points with coordinates

```
import matplotlib.pyplot as plt
DATA = ((1, 3),(2, 4), (3, 1), (4, 2))
dash_style = ((0, 20, −15, 30, 10), (1, 30, 0, 15, 10), (0, 40, 15, 15, 10), (1, 20, 30, 60, 10),)
fig, ax = plt.subplots()
(x,y) = zip(∗DATA)
ax.plot(x, y, marker='o')
for i in range(len(DATA)):
   (x,y) = DATA[i]
   (dd, dl, r, dr, dp) = dash_style[i]
   #print 'dashlen call', dl
   t = ax.text(x, y, str((x,y)), withdash=True, dashdirection=dd, dashlength=dl, rotation=r, →
      ↪ dashrotation=dr, dashpush=dp,)
ax.set_xlim((0.0, 5.0))
ax.set_ylim((0.0, 5.0))
plt.show()
```

yields:

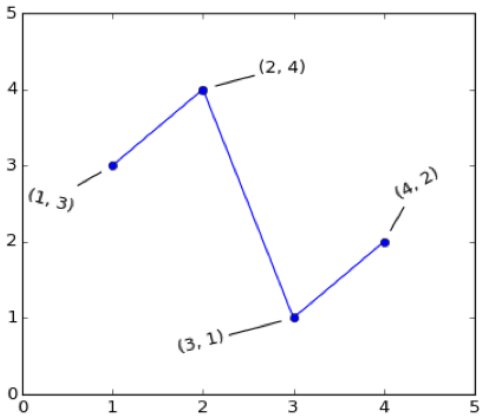

Figure 3.25 Points with coordinates

3.4.12 Date time plot

```
 import datetime
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import matplotlib.cbook as cbook
years    = mdates.YearLocator()   # every year
months   = mdates.MonthLocator()  # every month
yearsFmt = mdates.DateFormatter('%Y')
datafile = cbook.get_sample_data('goog.npy')
r = np.load(datafile).view(np.recarray)
fig, ax = plt.subplots()
ax.plot(r.date, r.adj_close)
```

```
ax.xaxis.set_major_locator(years)
ax.xaxis.set_major_formatter(yearsFmt)
ax.xaxis.set_minor_locator(months)
datemin = datetime.date(r.date.min().year, 1, 1)
datemax = datetime.date(r.date.max().year+1, 1, 1)
ax.set_xlim(datemin, datemax)
def price(x): return '$%1.2f'%x
ax.format_xdata = mdates.DateFormatter('%Y-%m-%d')
ax.format_ydata = price
ax.grid(True)
fig.autofmt_xdate()
plt.show()
```

yields:

Figure 3.26 Date time plot

3.4.13 Bar plot

```
 import matplotlib.pyplot as plt
import numpy as np
x,y = np.random.randn(2,100)
fig = plt.figure()
ax1 = fig.add_subplot(211)
ax1.xcorr(x, y, usevlines=True, maxlags=50, normed=True, lw=2)
ax1.grid(True)
ax1.axhline(0, color='black', lw=2)
ax2 = fig.add_subplot(212, sharex=ax1)
ax2.acorr(x, usevlines=True, normed=True, maxlags=50, lw=2)
ax2.grid(True)
ax2.axhline(0, color='black', lw=2)
plt.show()
```

yields:

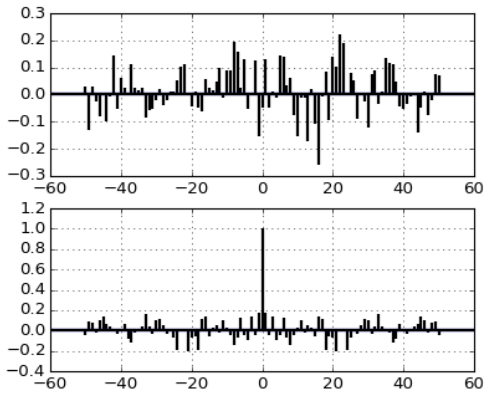

Figure 3.27 Bar plot

3.4.14 Multi-colored plot

```
 from mpl_toolkits.mplot3d import Axes3D
from matplotlib.collections import PolyCollection
from matplotlib.colors import colorConverter
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.gca(projection='3d')
cc = lambda arg: colorConverter.to_rgba(arg, alpha=0.6)
xs = np.arange(0, 10, 0.4)
verts = []
zs = [0.0, 1.0, 2.0, 3.0]
for z in zs:
    ys = np.random.rand(len(xs))
    ys[0], ys[−1] = 0, 0
    verts.append(list(zip(xs, ys)))
poly = PolyCollection(verts, facecolors = [cc('r'), cc('g'), cc('b'), cc('y')])
poly.set_alpha(0.7)
ax.add_collection3d(poly, zs=zs, zdir='y')
ax.set_xlabel('X')
ax.set_xlim3d(0, 10)
ax.set_ylabel('Y')
ax.set_ylim3d(−1, 4)
ax.set_zlabel('Z')
ax.set_zlim3d(0, 1)
plt.show()
```

yields:

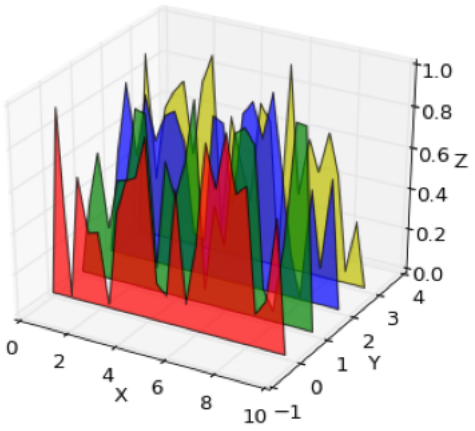

Figure 3.28 Multi-colored plot

3.4.15 Polar plot

```
import numpy as np
import matplotlib.pyplot as plt
r = np.arange(0, 3.0, 0.01)
theta = 2 * np.pi * r
ax = plt.subplot(111, polar=True)
ax.plot(theta, r, color='r', linewidth=3)
ax.set_rmax(2.0)
ax.grid(True)
ax.set_title("A line plot on a polar axis", va='bottom')
plt.show()
```

yields:

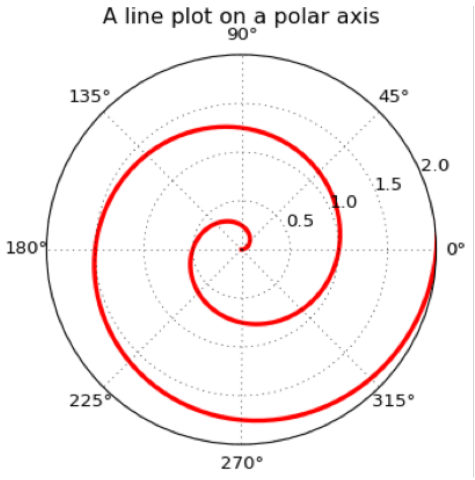

Figure 3.29 Polar plot

3.4.16 Another bar plot

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.path as path
fig, ax = plt.subplots()
data = np.random.randn(1000)
n, bins = np.histogram(data, 50)
left = np.array(bins[:-1])
right = np.array(bins[1:])
bottom = np.zeros(len(left))
top = bottom + n
XY = np.array([[left,left,right,right], [bottom,top,top,bottom]]).T
barpath = path.Path.make_compound_path_from_polys(XY)
patch = patches.PathPatch(barpath, facecolor='blue', edgecolor='gray', alpha=0.8)
ax.add_patch(patch)
ax.set_xlim(left[0], right[-1])
ax.set_ylim(bottom.min(), top.max())
plt.show()
```

yields:

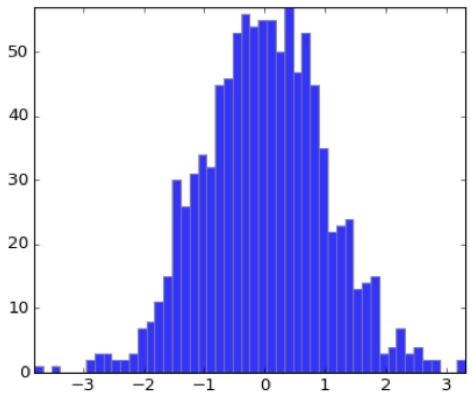

Figure 3.30 Another bar plot

3.4.17 Another histogram

```
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
mu = 100 # mean of distribution
sigma = 15 # standard deviation of distribution
x = mu + sigma * np.random.randn(10000)
num_bins = 50
n, bins, patches = plt.hist(x, num_bins, normed=1, facecolor='green', alpha=0.5)
y = mlab.normpdf(bins, mu, sigma)
plt.plot(bins, y, 'r--')
```

```
plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title(r'Histogram of IQ: $\mu=100$, $\sigma=15$')
plt.subplots_adjust(left=0.15)
plt.show()
```

yields:

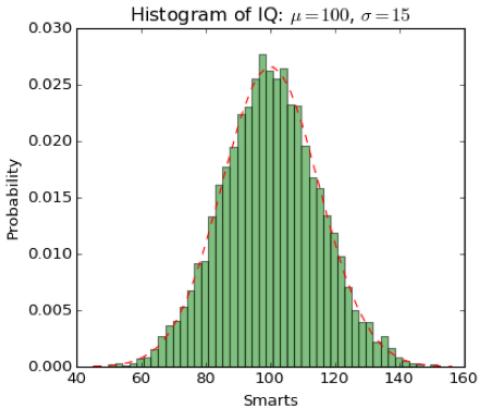

Figure 3.31 Another histogram

3.4.18 2D data plot

```
from pylab import *
x = randn(1000)
y = randn(1000)+5
hist2d(x,y,bins=40)
show()
```

yields:

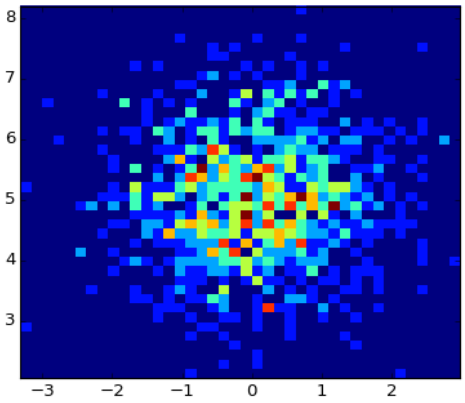

Figure 3.32 2D data plot

3.4.19 Vertical bar graph

```
import numpy as np
import matplotlib.pyplot as plt
N = 5
menMeans = (20, 35, 30, 35, 27)
menStd =   (2, 3, 4, 1, 2)
ind = np.arange(N)  # the x locations for the groups
width = 0.35       # the width of the bars
fig, ax = plt.subplots()
rects1 = ax.bar(ind, menMeans, width, color='r', yerr=menStd)
womenMeans = (25, 32, 34, 20, 25)
womenStd =   (3, 5, 2, 3, 3)
rects2 = ax.bar(ind+width, womenMeans, width, color='y', yerr=womenStd)
ax.set_ylabel('Scores')
ax.set_title('Scores by group and gender')
ax.set_xticks(ind+width)
ax.set_xticklabels( ('G1', 'G2', 'G3', 'G4', 'G5') )
ax.legend( (rects1[0], rects2[0]), ('Men', 'Women') )
def autolabel(rects):
   # attach some text labels
   for rect in rects:
      height = rect.get_height()
      ax.text(rect.get_x()+rect.get_width()/2., 1.05*height, '%d'%int(height),
            ha='center', va='bottom')
autolabel(rects1)
autolabel(rects2)
plt.show()
```

yields:

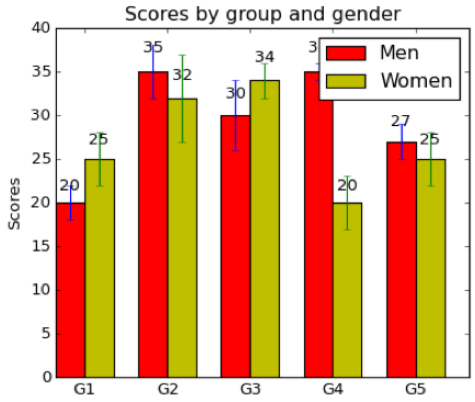

Figure 3.33 Vertical bar graph

3.4.20 Another vertical bar graph

```
import numpy as np
import matplotlib.pyplot as plt
N = 5
menMeans   = (20, 35, 30, 35, 27)
womenMeans = (25, 32, 34, 20, 25)
menStd     = (2, 3, 4, 1, 2)
womenStd   = (3, 5, 2, 3, 3)
ind = np.arange(N)
width = 0.35
p1 = plt.bar(ind, menMeans,   width, color='r', yerr=womenStd)
p2 = plt.bar(ind, womenMeans, width, color='y',bottom=menMeans, yerr=menStd)
plt.ylabel('Scores')
plt.title('Scores by group and gender')
plt.xticks(ind+width/2., ('G1', 'G2', 'G3', 'G4', 'G5') )
plt.yticks(np.arange(0,81,10))
plt.legend( (p1[0], p2[0]), ('Men', 'Women') )
plt.show()
```

yields:

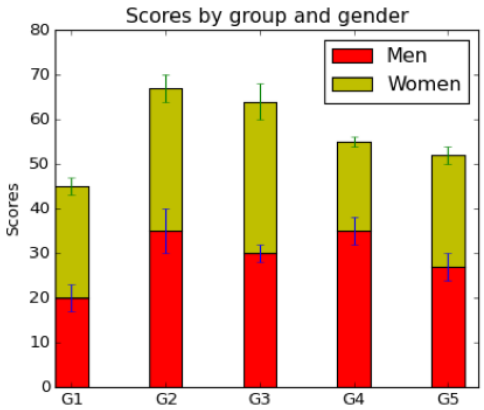

Figure 3.34 Another vertical bar graph

3.4.21 Horizontal bar graph

```
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt
people = ('Tom', 'Dick', 'Harry', 'Slim', 'Jim')
y_pos = np.arange(len(people))
performance = 3 + 10 * np.random.rand(len(people))
error = np.random.rand(len(people))
plt.barh(y_pos, performance, xerr=error, align='center', alpha=0.4)
plt.yticks(y_pos, people)
plt.xlabel('Performance')
plt.title('How fast do you want to go today?')
plt.show()
```

yields:

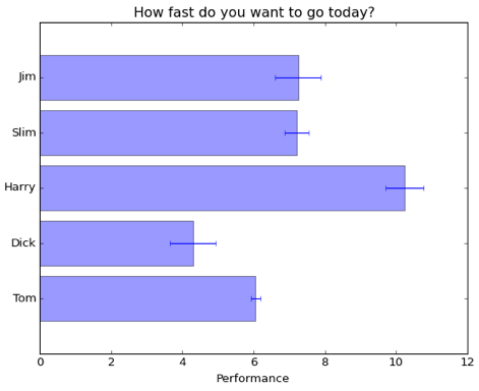

Figure 3.35 Horizontal bar graph

3.4.22 3D bar graph

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x, y = np.random.rand(2, 100) * 4
hist, xedges, yedges = np.histogram2d(x, y, bins=4)
elements = (len(xedges) - 1) * (len(yedges) - 1)
xpos, ypos = np.meshgrid(xedges[:-1]+0.25, yedges[:-1]+0.25)
xpos = xpos.flatten()
ypos = ypos.flatten()
zpos = np.zeros(elements)
dx = 0.5 * np.ones_like(zpos)
dy = dx.copy()
dz = hist.flatten()
ax.bar3d(xpos, ypos, zpos, dx, dy, dz, color='b', zsort='average')
plt.show()
```

yields:

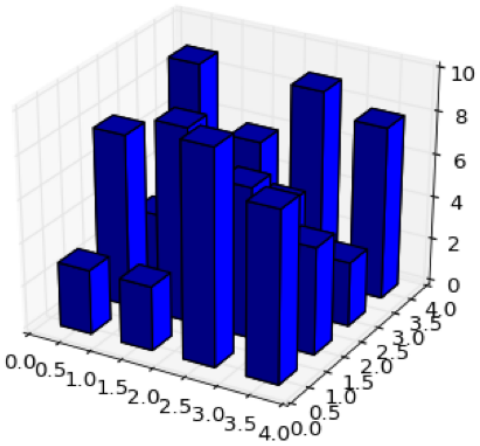

Figure 3.36 3D bar graph

3.4.23 Multi-colored bar plots

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
for c, z in zip(['r', 'g', 'b', 'y'], [30, 20, 10, 0]):
    xs = np.arange(20)
    ys = np.random.rand(20)
    cs = [c] * len(xs)
```

```
    cs[0] = 'c'
    ax.bar(xs, ys, zs=z, zdir='y', color=cs, alpha=0.8)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
```

yields:

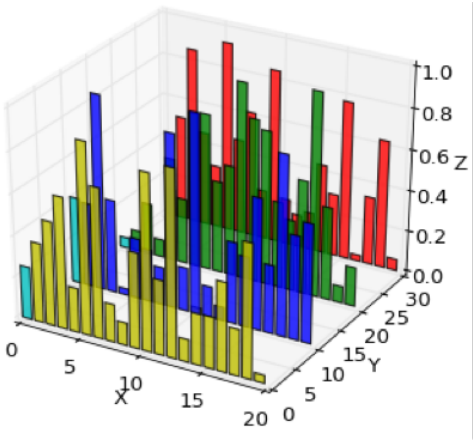

Figure 3.37 Multi-colored bar plots

3.4.24 Contours

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import cm
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
X, Y, Z = axes3d.get_test_data(0.05)
cset = ax.contour(X, Y, Z, cmap=cm.coolwarm)
ax.clabel(cset, fontsize=9, inline=1)
plt.show()
```

yields:

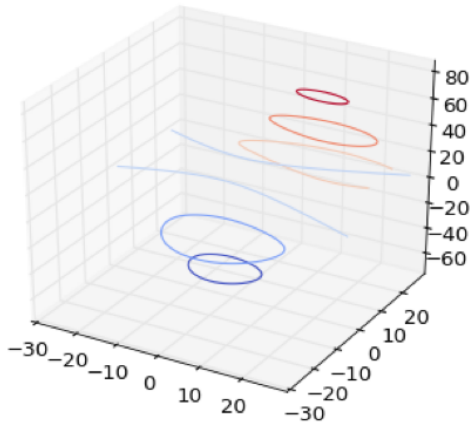

Figure 3.38 Contours

3.4.25 Advanced contours

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import cm

fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y, Z = axes3d.get_test_data(0.05)
ax.plot_surface(X, Y, Z, rstride=8, cstride=8, alpha=0.3)
cset = ax.contour(X, Y, Z, zdir='z', offset=-100, cmap=cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='x', offset=-40, cmap=cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='y', offset=40, cmap=cm.coolwarm)

ax.set_xlabel('X')
ax.set_xlim(-40, 40)
ax.set_ylabel('Y')
ax.set_ylim(-40, 40)
ax.set_zlabel('Z')
ax.set_zlim(-100, 100)

plt.show()
```

yields:

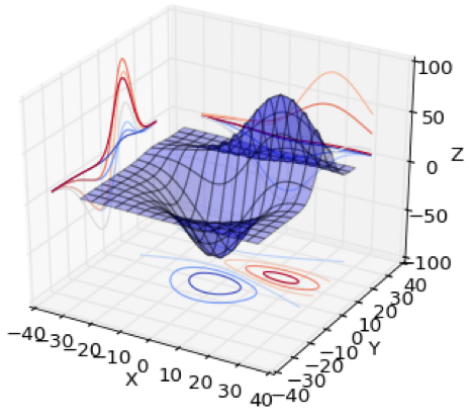

Figure 3.39 Advanced contours

3.4.26 Surface

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
plt.ion()
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
X, Y, Z = axes3d.get_test_data(0.1)
ax.plot_wireframe(X, Y, Z, rstride=5, cstride=5)
for angle in range(0, 360):
    ax.view_init(30, angle)
    plt.draw()
```

yields:

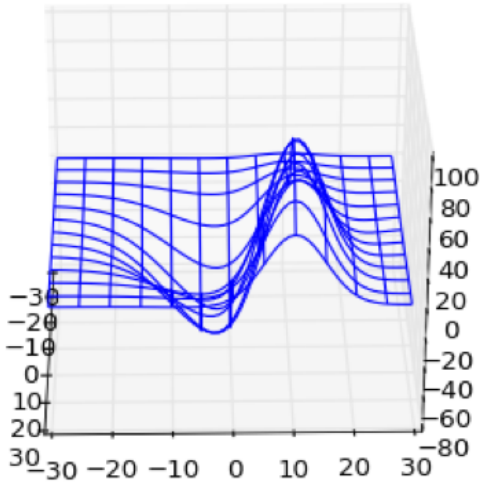

Figure 3.40 Surface

3.4.27 Colored surface

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.gca(projection='3d')
X = np.arange(−5, 5, 0.25)
Y = np.arange(−5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm,
    linewidth=0, antialiased=False)
ax.set_zlim(−1.01, 1.01)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.show()
```

yields:

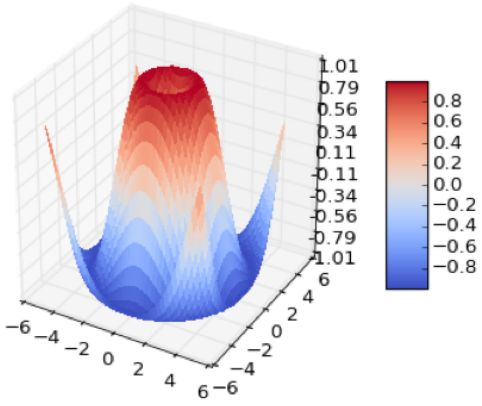

Figure 3.41 Colored surface

3.4.28 Another colored surface

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib.pyplot as plt
import numpy as np
n_angles = 36
n_radii = 8
radii = np.linspace(0.125, 1.0, n_radii)
angles = np.linspace(0, 2*np.pi, n_angles, endpoint=False)
angles = np.repeat(angles[...,np.newaxis], n_radii, axis=1)
x = np.append(0, (radii*np.cos(angles)).flatten())
y = np.append(0, (radii*np.sin(angles)).flatten())
z = np.sin(-x*y)
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_trisurf(x, y, z, cmap=cm.jet, linewidth=0.2)
plt.show()
```

yields:

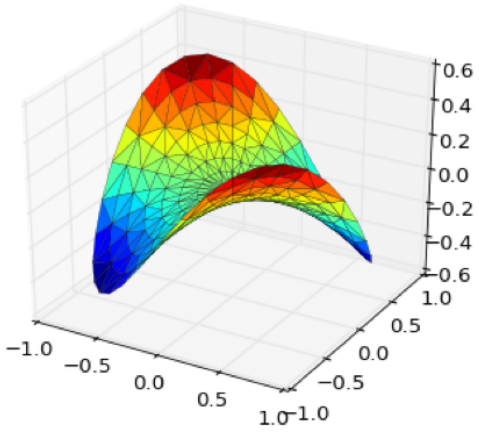

Figure 3.42 Another colored surface

3.4.29 Points and contours

```
from numpy.random import uniform, seed
from matplotlib.mlab import griddata
import matplotlib.pyplot as plt
import numpy as np
seed(0)
npts = 200
x = uniform(−2, 2, npts)
y = uniform(−2, 2, npts)
z = x∗np.exp(−x∗∗2 − y∗∗2)
xi = np.linspace(−2.1, 2.1, 100)
yi = np.linspace(−2.1, 2.1, 200)
zi = griddata(x, y, z, xi, yi, interp='linear')
CS = plt.contour(xi, yi, zi, 15, linewidths=0.5, colors='k')
CS = plt.contourf(xi, yi, zi, 15, cmap=plt.cm.rainbow, vmax=abs(zi).max(), vmin=−abs(zi).max →
    ↪ ())
plt.colorbar()  # draw colorbar
plt.scatter(x, y, marker='o', c='b', s=5, zorder=10)
plt.xlim(−2, 2)
plt.ylim(−2, 2)
plt.title('griddata test (%d points)' % npts)
plt.show()
```

yields:

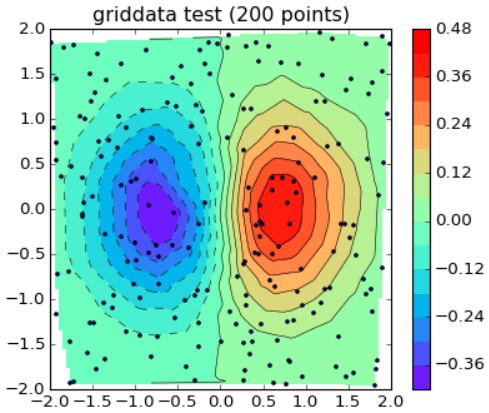

Figure 3.43 Points and contours

3.4.30 Points and lines

```
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.gca(projection='3d')

x = np.linspace(0, 1, 100)
y = np.sin(x * 2 * np.pi) / 2 + 0.5
ax.plot(x, y, zs=0, zdir='z', label='zs=0, zdir=z')
colors = ('r', 'g', 'b', 'k')
for c in colors:
    x = np.random.sample(20)
    y = np.random.sample(20)
    ax.scatter(x, y, 0, zdir='y', c=c)
ax.legend()
ax.set_xlim3d(0, 1)
ax.set_ylim3d(0, 1)
ax.set_zlim3d(0, 1)
plt.show()
```

yields:

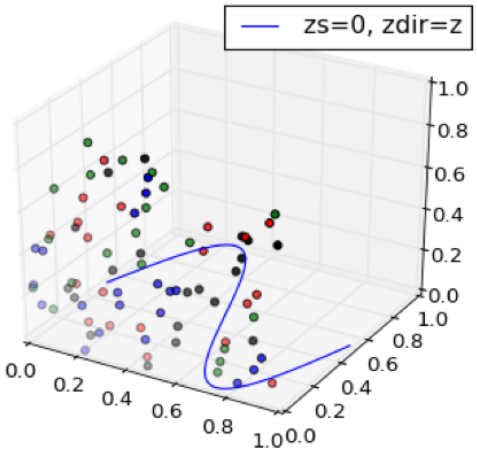

Figure 3.44 Points and lines

3.4.31 Points

```
import matplotlib.pyplot as plt
from numpy.random import rand
for color in ['red', 'green', 'blue']:
    n = 750
    x, y = rand(2, n)
    scale = 200.0 * rand(n)
    plt.scatter(x, y, c=color, s=scale, label=color,
                alpha=0.3, edgecolors='none')
plt.legend()
plt.grid(True)
plt.show()
```

yields:

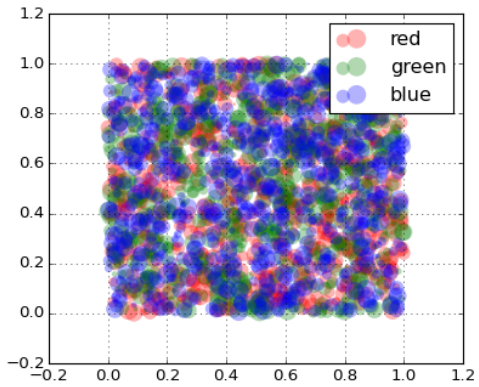

Figure 3.45 Points

3.4.32 Complex plot

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
x = np.random.randn(1000)
y = np.random.randn(1000)
fig, axScatter = plt.subplots(figsize=(5.5,5.5))
axScatter.scatter(x, y)
axScatter.set_aspect(1.)
divider = make_axes_locatable(axScatter)
axHistx = divider.append_axes("top", 1.2, pad=0.1, sharex=axScatter)
axHisty = divider.append_axes("right", 1.2, pad=0.1, sharey=axScatter)
plt.setp(axHistx.get_xticklabels() + axHisty.get_yticklabels(),visible=False)
binwidth = 0.25
xymax = np.max( [np.max(np.fabs(x)), np.max(np.fabs(y))] )
lim = ( int(xymax/binwidth) + 1) * binwidth
bins = np.arange(-lim, lim + binwidth, binwidth)
axHistx.hist(x, bins=bins)
axHisty.hist(y, bins=bins, orientation='horizontal')
for tl in axHistx.get_xticklabels():
    tl.set_visible(False)
axHistx.set_yticks([0, 50, 100])
for tl in axHisty.get_yticklabels():
    tl.set_visible(False)
axHisty.set_xticks([0, 50, 100])
plt.draw()
plt.show()
```

yields:

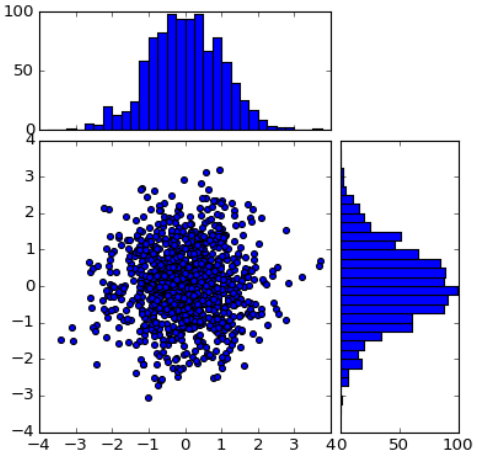

Figure 3.46 Complex plot

3.4.33 Pie chart

```
import matplotlib.pyplot as plt
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
sizes = [15, 30, 45, 10]
colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral']
explode = (0, 0.1, 0, 0)
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow= →
    ↪ True, startangle=90)
plt.axis('equal')
plt.show()
```

yields:

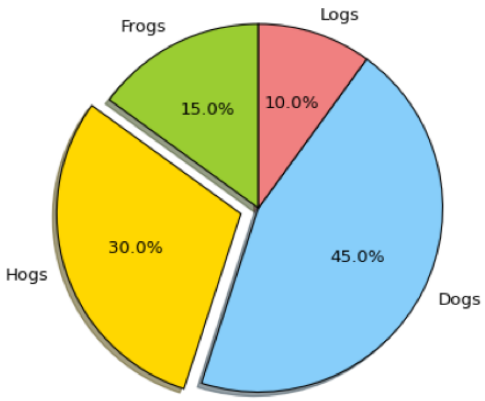

Figure 3.47 Pie chart

3.4.34 Radar

```
import numpy as np
import matplotlib.pyplot as plt
N = 20
theta = np.linspace(0.0, 2 * np.pi, N, endpoint=False)
radii = 10 * np.random.rand(N)
width = np.pi / 4 * np.random.rand(N)
ax = plt.subplot(111, polar=True)
bars = ax.bar(theta, radii, width=width, bottom=0.0)
for r, bar in zip(radii, bars):
    bar.set_facecolor(plt.cm.jet(r / 10.))
    bar.set_alpha(0.5)
plt.show()
```

yields:

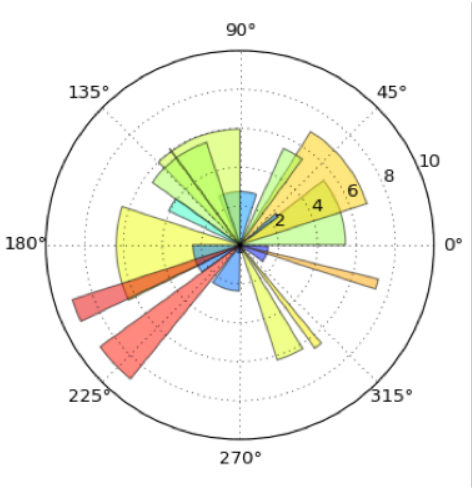

Figure 3.48 Radar

3.4.35 Another radar

```
import numpy as np
import matplotlib.pyplot as plt
N = 150
r = 2 * np.random.rand(N)
theta = 2 * np.pi * np.random.rand(N)
area = 200 * r**2 * np.random.rand(N)
colors = theta
ax = plt.subplot(111, polar=True)
c = plt.scatter(theta, r, c=colors, s=area, cmap=plt.cm.hsv)
c.set_alpha(0.75)
plt.show()
```

yields:

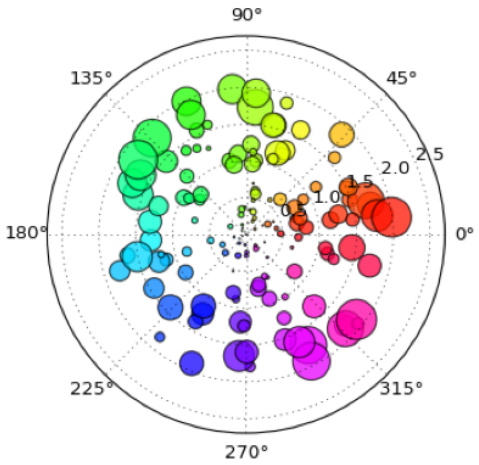

Figure 3.49 Another radar

3.4.36 Another complex radar

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.path import Path
from matplotlib.spines import Spine
from matplotlib.projections.polar import PolarAxes
from matplotlib.projections import register_projection

def _radar_factory(num_vars):
    theta = 2*np.pi * np.linspace(0, 1-1./num_vars, num_vars)
    theta += np.pi/2
    def unit_poly_verts(theta):
        x0, y0, r = [0.5] * 3
        verts = [(r*np.cos(t) + x0, r*np.sin(t) + y0) for t in theta]
        return verts
    class RadarAxes(PolarAxes):
        name = 'radar'
        def fill(self, *args, **kwargs):
            closed = kwargs.pop('closed', True)
            return super(RadarAxes, self).fill(closed=closed, *args, **kwargs)
    def plot(self, *args, **kwargs):
            lines = super(RadarAxes, self).plot(*args, **kwargs)
            for line in lines:
                self._close_line(line)
     def _close_line(self, line):
        x, y = line.get_data()
        if x[0] != x[-1]:
            x = np.concatenate((x, [x[0]]))
            y = np.concatenate((y, [y[0]]))
            line.set_data(x, y)
    def set_varlabels(self, labels):
        self.set_thetagrids(theta * 180/np.pi, labels)
    def _gen_axes_patch(self):
        verts = unit_poly_verts(theta)
```

```
        return plt.Polygon(verts, closed=True, edgecolor='k')
    def _gen_axes_spines(self):
        spine_type = 'circle'
        verts = unit_poly_verts(theta)
        verts.append(verts[0])
        path = Path(verts)
        spine = Spine(self, spine_type, path)
        spine.set_transform(self.transAxes)
        return {'polar': spine}
register_projection(RadarAxes)
return theta

labels = ['v1', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7', 'v8', 'v9']
values = [1, 1, 2, 7, 4, 0, 3, 10, 6]
optimum = [5, 3, 2, 4, 5, 7, 5, 8, 5]
N = len(labels)
theta = _radar_factory(N)
max_val = max(max(optimum), max(values))
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1, projection='radar')
ax.plot(theta, values, color='k')
ax.plot(theta, optimum, color='r')
ax.set_varlabels(labels)
plt.show()
```

yields:

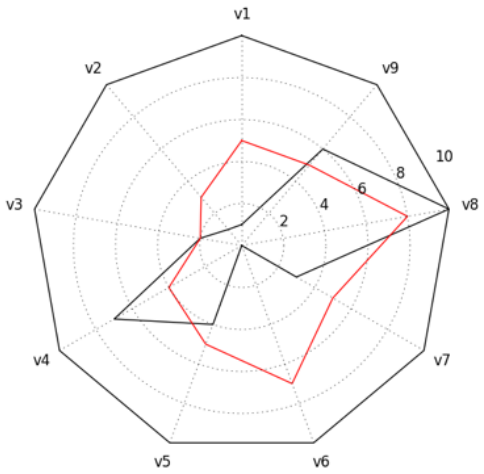

Figure 3.50 Another complex radar

3.4.37 3 Sankey diagrams

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.sankey import Sankey
```

```
Sankey(flows=[0.25, 0.15, 0.60, −0.20, −0.15, −0.05, −0.50, −0.10],
      labels=['', '', '', 'First', 'Second', 'Third', 'Fourth', 'Fifth'],
      orientations=[−1, 1, 0, 1, 1, 1, 0, −1]).finish()
plt.title("The default settings produce a diagram like this.")

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1, xticks=[], yticks=[],
                title="Flow Diagram of a Widget")
sankey = Sankey(ax=ax, scale=0.01, offset=0.2, head_angle=180,
            format='%.0f', unit='%')
sankey.add(flows=[25, 0, 60, −10, −20, −5, −15, −10, −40],
        labels = ['', '', '', 'First', 'Second', 'Third', 'Fourth',
              'Fifth', 'Hurray!'],
        orientations=[−1, 1, 0, 1, 1, 1, −1, −1, 0],
        pathlengths = [0.25, 0.25, 0.25, 0.25, 0.25, 0.6, 0.25, 0.25,
                  0.25],
        patchlabel="Widget\nA",
        alpha=0.2, lw=2.0) # Arguments to matplotlib.patches.PathPatch()
diagrams = sankey.finish()
diagrams[0].patch.set_facecolor('#37c959')
diagrams[0].texts[−1].set_color('r')
diagrams[0].text.set_fontweight('bold')

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1, xticks=[], yticks=[], title="Two Systems")
flows = [0.25, 0.15, 0.60, −0.10, −0.05, −0.25, −0.15, −0.10, −0.35]
sankey = Sankey(ax=ax, unit=None)
sankey.add(flows=flows, label='one',
        orientations=[−1, 1, 0, 1, 1, 1, −1, −1, 0])
sankey.add(flows=[−0.25, 0.15, 0.1], fc='#37c959', label='two',
        orientations=[−1, −1, −1], prior=0, connect=(0, 0))
diagrams = sankey.finish()
diagrams[−1].patch.set_hatch('/')
plt.legend(loc='best')
plt.show()
```

yields:

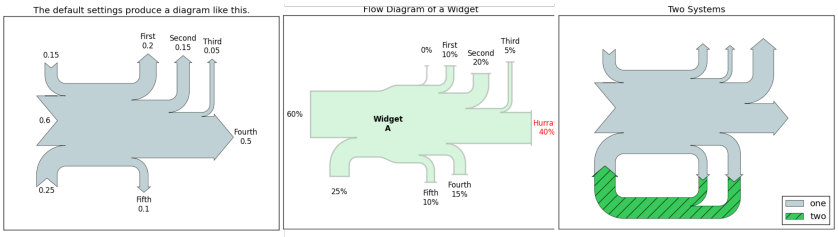

Figure 3.51 3 Sankey diagrams

3.4.38 Vectors

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.gca(projection='3d')
x, y, z = np.meshgrid(np.arange(−0.8, 1, 0.2), np.arange(−0.8, 1, 0.2), np.arange(−0.8, 1, 0.8))
u = np.sin(np.pi ∗ x) ∗ np.cos(np.pi ∗ y) ∗ np.cos(np.pi ∗ z)
v = −np.cos(np.pi ∗ x) ∗ np.sin(np.pi ∗ y) ∗ np.cos(np.pi ∗ z)
w = (np.sqrt(2.0 / 3.0) ∗ np.cos(np.pi ∗ x) ∗ np.cos(np.pi ∗ y) ∗
    np.sin(np.pi ∗ z))
ax.quiver(x, y, z, u, v, w, length=0.1)
plt.show()
```

yields:

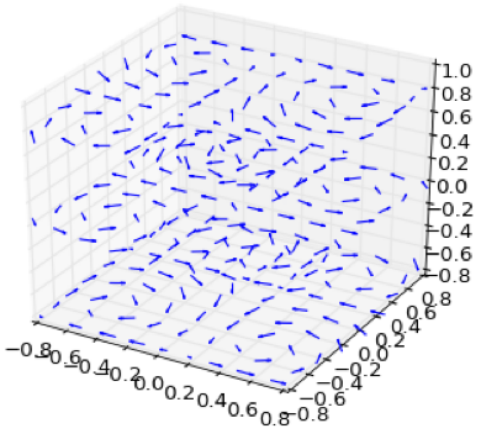

Figure 3.52 Vectors

3.4.39 Insets

```
from pylab import ∗
dt = 0.001
t = arange(0.0, 10.0, dt)
r = exp(−t[:1000]/0.05)
x = randn(len(t))
s = convolve(x,r)[:len(x)]∗dt
plot(t, s)
axis([0, 1, 1.1∗amin(s), 2∗amax(s) ])
xlabel('time (s)')
ylabel('current (nA)')
title('Gaussian colored noise')
a = axes([.65, .6, .2, .2], axisbg='y')
n, bins, patches = hist(s, 400, normed=1)
title('Probability')
setp(a, xticks=[], yticks=[])
a = axes([0.2, 0.6, .2, .2], axisbg='y')
```

```
plot(t[:len(r)], r)
title('Impulse response')
setp(a, xlim=(0,.2), xticks=[], yticks=[])
show()
```

yields;

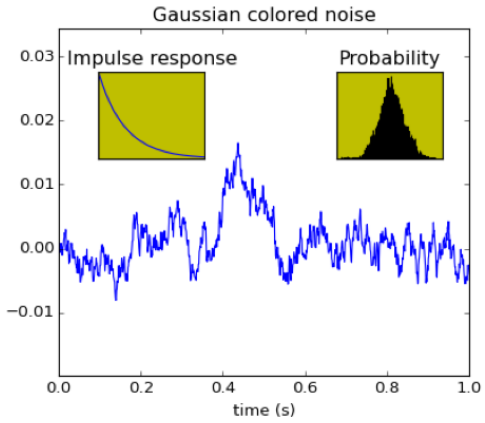

Figure 3.53 Insets

Exercise 3.4 Model fitting

Consider the following measurements: $x = [-5,-4,-3,-2,-1,0,1,2,3,4,5]$ and $y = [12.5,7,5,1.6,-1.1,-3.2,-2.6,-0.7,3.4,7.6,13]$.
Plot the points $y = f(x)$ in blue circles, with lines between consecutive points.
The model template $y = \alpha x^2 + \beta$ is used. Following a root mean square approach, compute the best values for α and β.
Plot the resulting curve $y = \alpha x^2 + \beta$ with 41 points between -5 and 5.

Solution:

$$\forall i, y_i = \alpha x_i^2 + \beta$$
$$\forall i, y_i = \begin{bmatrix}\alpha & \beta\end{bmatrix} \begin{bmatrix} x_i^2 \\ 1 \end{bmatrix}$$
$$Y = [y_0 \dots y_{n-1}]$$
$$\Theta = \begin{bmatrix}\alpha & \beta\end{bmatrix}$$
$$M_X = \begin{bmatrix} x_0^2 & \cdots & x_{n-1}^2 \\ 1 & \dots & 1 \end{bmatrix} \text{ with } \hat{Y} = \Theta M_X$$

It yields:

$$\hat{\Theta} = \hat{Y} M_X^T (M_X X_X^T)^{-1}$$

then

$$\hat{Y}\prime = \hat{\Theta} M_{X\prime}$$

```
import numpy as np
import matplotlib.pyplot as plt

x=np.r_[−5:6]
y=[12.5,7,5,1.6,−1.1,−3.2,−2.6,−0.7,3.4,7.6,13]

Y=np.matrix(y)
M=np.vstack([np.power(np.matrix(x),2),np.ones(Y.shape)])
theta=Y∗M.transpose()∗np.linalg.inv(M∗M.transpose())
print(theta)
xest=np.linspace(−5,5,41)
Mest=np.vstack([np.power(np.matrix(xest),2), np.ones((1,len(xest)))])
Yest=theta∗Mest
yest=Yest.tolist()[0]

plt.plot(x,y,'b')
plt.plot(x,y,'bo')
plt.plot(xest,yest,'r:')
plt.grid()
plt.show()
```

Exercise 3.5 Outlier

Generate a dataset with an outlier:
x=rand(1,100)
x[0,40]+=2
x[27]+=3
x[51]−=2

Plot the dataset
Use a moving windows of 11 consecutive values to detect the outlier. If $| \nu - \mu | > 4\sigma$, there is a detected outlier, with ν the central value of the window, μ the average value of the window without central value and σ the standard deviation of the window without central value.
Add to the plotted dataset the detected outliers.

Solution:

```
import numpy as np
import matplotlib.pyplot as plt
__all__=['findOutliers']
def _isOutlier(x:list,midlleindex:int,halfwidth=5,threshold=4):
    window=np.hstack([x[midlleindex−halfwidth:midlleindex],x[midlleindex+1: →
        ↪ midlleindex+halfwidth+1]])
    windowmean=np.mean(window)
    windowstd=np.std(window)
    if abs(x[midlleindex]−windowmean)>threshold∗windowstd:
        return True
    return False

def findOutliers(x:list):
```

```
    outliers=[_isOutlier(x,i) for i in range(5,100-5)]
    outlierIndice=[i+5 for i in np.nonzero(outliers)]
    outlierValues=[x[i] for i in outlierIndice]
    return outlierIndice,outlierValues

if __name__ == '__main__':
    x=np.random.rand(1,100).reshape(-1)
    x[40]+=2
    x[27]+=3
    x[51]-=2
    (outlierIndice,outlierValues)=findOutliers(x)
    print(outlierIndice)
    plt.plot(x)
    plt.plot(outlierIndice,outlierValues,'ro')
    plt.grid()
    plt.axis('tight')
    plt.show()
```

3.5 Code for calculating entropy and information gain

```
__author__ = 'amayrim'
# calculate entropy and information gain
# calculate entropy and information gain
__all__ = []
import sqlite3
from statistics import stdev
from time import strftime, localtime, strptime, mktime
from matplotlib.pylab import *

def to_epochtime(datetime):  # epoch time in seconds
    epochdate=mktime(strptime(datetime, '%d/%m/%Y %H:%M:%S'))
    return int(epochdate)


def to_datetime(epochtime):  # epoch time in seconds
    return strftime('%d/%m/%Y %H:%M:%S', localtime(int(round(epochtime, 0))))


def to_timequantum(epochtime, seconds_in_timequantum: int):# epoch time in seconds
    return (epochtime // seconds_in_timequantum) * seconds_in_timequantum


class DatabaseExtraction():

    def __init__(self, database_name: str, starting_datetime: '%d/%m/%Y %H:%M:%S', →
        ↪ ending_datetime: '%d/%m/%Y %H:%M:%S', seconds_in_sample: int=1800):
        self.connection = sqlite3.connect(database_name)
        self.starting_datetime = starting_datetime
        self.ending_datetime = ending_datetime
        self.seconds_in_timequantum = seconds_in_sample
```

```
def get_seconds_in_sample(self):
    return self.seconds_in_timequantum

def get_raw_measurement(self, variable_name: str, remove_outliers: bool=True, →
    ↪ outlier_sensitivity: float=5):
    cursor = self.connection.cursor()
    cursor.execute('SELECT time, val FROM %s WHERE time>=%i AND time →
    ↪ <%i ORDER BY time ASC' % (variable_name, to_epochtime(self. →
    ↪ starting_datetime) * 1000, to_epochtime(self.ending_datetime) * 1000))
    epochtimes, values = [], []
    for record in cursor.fetchall():
        epochtimes.append(int(record[0]) // 1000)
        values.append(record[1])
    if remove_outliers:
        delta_values = [values[k + 1] - values[k] for k in range(len(values)-1)]
        mean_delta = mean(delta_values)
        sigma_delta = stdev(delta_values)
        for k in range(1,len(values)-1):
            pdiff = values[k] - values[k-1]
            fdiff = values[k+1] - values[k]
            if (abs(pdiff) > mean_delta + outlier_sensitivity * sigma_delta) and (abs(fdiff) > →
    ↪ mean_delta + outlier_sensitivity * sigma_delta) and pdiff * fdiff > 0:
                print('Outlier detected in %s at time %s: %f (replaced by →
    ↪ average value)' % (variable_name, to_datetime(epochtimes[k]), values[k]))
                values[k] = (values[k-1] + values[k+1])/2
    return epochtimes, values

def get_counter_measurement(self, variable_name: str):
    data_epochtimes, data_values = self.get_raw_measurement(variable_name, remove_outliers →
    ↪ =False)
    counters_dict = dict()
    for i in range(len(data_epochtimes)):
        if data_values[i] == 1:
            datasample = self.to_datasample_epochtime(data_epochtimes[i])
            if datasample in counters_dict:
                counters_dict[datasample] += 1
            else:
                counters_dict[datasample] = 1
    sampled_data_epochtimes = [datasample for datasample in range(self. →
    ↪ to_datasample_epochtime(to_epochtime(self.starting_datetime)), self. →
    ↪ to_datasample_epochtime(to_epochtime(self.ending_datetime)), self. →
    ↪ seconds_in_timequantum)]
    sampled_data_values = []
    for i in range(len(sampled_data_epochtimes)):
        if sampled_data_epochtimes[i] in counters_dict:
            sampled_data_values.append(counters_dict[sampled_data_epochtimes[i]])
        else:
            sampled_data_values.append(0)
    return sampled_data_epochtimes, sampled_data_values


def get_sampled_measurement(self, variable_name: str, remove_outliers: bool=True, →
    ↪ outlier_sensitivity: float=5):
```

```
        data_epochtimes, data_values = self.get_raw_measurement(variable_name, remove_outliers →
        ↪ , outlier_sensitivity)
        if len(data_epochtimes) == 0:
            return None
        augmented_data_epochtimes = [self.to_datasample_epochtime(to_epochtime(self. →
        ↪ starting_datetime))]
        augmented_data_values = [data_values[0]]
        for i in range(len(data_epochtimes)):
            data_epochtime = data_epochtimes[i]
            for epochtime in range(self.to_datasample_epochtime(augmented_data_epochtimes[-1]) →
        ↪ + self.seconds_in_timequantum, self.to_datasample_epochtime(data_epochtime), self. →
        ↪ seconds_in_timequantum):
                augmented_data_epochtimes.append(epochtime)
                augmented_data_values.append(augmented_data_values[-1])
            if self.to_datasample_epochtime(data_epochtime) > self.to_datasample_epochtime( →
        ↪ augmented_data_epochtimes[-1]):
                augmented_data_epochtimes.append(self.to_datasample_epochtime(data_epochtime))
                augmented_data_values.append(augmented_data_values[-1])
            augmented_data_epochtimes.append(data_epochtime)
            augmented_data_values.append(data_values[i])
        for epochtime in range(self.to_datasample_epochtime(augmented_data_epochtimes[-1]), →
        ↪ to_epochtime(self.ending_datetime), self.seconds_in_timequantum):
            augmented_data_epochtimes.append(epochtime + self.seconds_in_timequantum)
            augmented_data_values.append(augmented_data_values[-1])
        sampled_data_epochtimes, sampled_data_values = [], []
        integrator = 0
        for i in range(1,len(augmented_data_epochtimes)):
            previous_epochtime = augmented_data_epochtimes[i - 1]
            previous_datasample_epochtime = self.to_datasample_epochtime( →
        ↪ augmented_data_epochtimes[i-1])
            epochtime = augmented_data_epochtimes[i]
            datasample_epochtime = self.to_datasample_epochtime(augmented_data_epochtimes[i])
            previous_value = augmented_data_values[i - 1]
            integrator += (epochtime - previous_epochtime) * previous_value
            if datasample_epochtime > previous_datasample_epochtime:
                sampled_data_epochtimes.append(previous_datasample_epochtime)
                sampled_data_values.append(integrator / self.seconds_in_timequantum)
                integrator = 0
        return sampled_data_epochtimes, sampled_data_values

    def to_datasample_epochtime(self, epochtime):
        return (epochtime // self.seconds_in_timequantum) * self.seconds_in_timequantum

    def close(self):
        self.connection.commit()
        self.connection.close()

def entropy(variable_data): #occupancy is the target
    count0, count1 = 0, 0  # counter for the class0 and class1 in the occupancy column
    for i in range(len(variable_data)):
        if variable_data[i] == 0: count0 += 1
        else:  count1 += 1
    p0, p1 = count0 / len(variable_data), count1 / len(variable_data)
    v0 = math.log(p0) / math.log(2) if p0 != 0 else 0  # to avoid log(0)
    v1 = math.log(p1) / math.log(2) if p1 != 0 else 0  # to avoid log(0)
```

```
    return − p0 ∗ v0 − p1 ∗ v1

def information_gain(feature_data, occupancy_data):
  occupancy_data0, occupancy_data1, feature_data0, feature_data1 = [], [], [], []
  for i in range(len(occupancy_data)):
    if feature_data[i] == 0:
      feature_data0.append(feature_data[i])
      occupancy_data0.append(occupancy_data[i])
    else:
      feature_data1.append(feature_data[i])
      occupancy_data1.append(occupancy_data[i])
  entropy1, entropy2 = entropy(occupancy_data0), entropy(occupancy_data1)
  return entropy(occupancy_data) − (len(feature_data0) / len(occupancy_data)) ∗ entropy1 − (len →
      ↪ (feature_data1) / len(occupancy_data)) ∗ entropy2

def discretize(value, bounds=((0,0), (0,1), (1,1))): # discretization of window and door data
    for i in range(len(bounds)):
      if value >= bounds[i][0] and value<= bounds[i][1]:
        return i
    return −1

seconds_in_sample = 1800
database_extraction = DatabaseExtraction('testing.db', '17/05/2015 00:30:00', ' →
    ↪ 21/05/2015 00:30:00', seconds_in_sample)
#################################
times, occupancies = database_extraction.get_sampled_measurement('labels')
times, door_openings = database_extraction.get_sampled_measurement('Door_contact')
times, window_openings = database_extraction.get_sampled_measurement('window')

discretized_occupancies = [discretize(occupancy, ((0, 0.5), (0.5, 6))) for occupancy in →
    ↪ occupancies]
discretized_door_openings = [discretize(door_opening, ((0,0), (0,1), (1,1))) for door_opening in →
    ↪ door_openings]
discretized_window_openings = [discretize(window_opening, ((0,0), (0,1), (1,1))) for →
    ↪ window_opening in window_openings]

print("1-information gain for door position=", information_gain( →
    ↪ discretized_door_openings, discretized_occupancies))
print("2-information gain for window position=", information_gain( →
    ↪ discretized_window_openings, discretized_occupancies))
```

3.6 Code for estimating occupancy using decision tree

```
# example of scikit learn
__author__ = 'amayrim'
__all__ = []
import sqlite3
```

```
import numpy
from statistics import stdev
from time import strftime, localtime, strptime, mktime
from matplotlib.pylab import *
from sklearn import metrics, tree


def to_epochtime(datetime):  # epoch time in seconds
    epochdate=mktime(strptime(datetime, '%d/%m/%Y %H:%M:%S'))
    return int(epochdate)


def to_datetime(epochtime):  # epoch time in seconds
    return strftime('%d/%m/%Y %H:%M:%S', localtime(int(round(epochtime, 0))))


def to_timequantum(epochtime, seconds_in_timequantum: int):# epoch time in seconds
    return (epochtime // seconds_in_timequantum) * seconds_in_timequantum


class DatabaseExtraction():

    def __init__(self, database_name: str, starting_datetime: '%d/%m/%Y %H:%M:%S', →
     ↪ ending_datetime: '%d/%m/%Y %H:%M:%S', seconds_in_sample: int=1800):
        self.connection = sqlite3.connect(database_name)
        self.starting_datetime = starting_datetime
        self.ending_datetime = ending_datetime
        self.seconds_in_timequantum = seconds_in_sample

    def get_seconds_in_sample(self):
        return self.seconds_in_timequantum

    def get_raw_measurement(self, variable_name: str, remove_outliers: bool=True, →
     ↪ outlier_sensitivity: float=5):
        cursor = self.connection.cursor()
        cursor.execute('SELECT time, val FROM %s WHERE time>=%i AND time →
     ↪ <%i ORDER BY time ASC' % (variable_name, to_epochtime(self. →
     ↪ starting_datetime) * 1000, to_epochtime(self.ending_datetime) * 1000))
        epochtimes, values = [], []
        for record in cursor.fetchall():
            epochtimes.append(int(record[0]) // 1000)
            values.append(record[1])
        if remove_outliers:
            delta_values = [values[k + 1] - values[k] for k in range(len(values)-1)]
            mean_delta = mean(delta_values)
            sigma_delta = stdev(delta_values)
            for k in range(1,len(values)-1):
                pdiff = values[k] - values[k-1]
                fdiff = values[k+1] - values[k]
                if (abs(pdiff) > mean_delta + outlier_sensitivity * sigma_delta) and (abs(fdiff) > →
     ↪ mean_delta + outlier_sensitivity * sigma_delta) and pdiff * fdiff > 0:
                    print('Outlier detected in %s at time %s: %f (replaced by →
     ↪  average value)' % (variable_name, to_datetime(epochtimes[k]), values[k]))
                    values[k] = (values[k-1] + values[k+1])/2
        return epochtimes, values
```

```
def get_counter_measurement(self, variable_name: str):
    data_epochtimes, data_values = self.get_raw_measurement(variable_name, →
↪ remove_outliers=False)
    counters_dict = dict()
    for i in range(len(data_epochtimes)):
        if data_values[i] == 1:
            datasample = self.to_datasample_epochtime(data_epochtimes[i])
            if datasample in counters_dict:
                counters_dict[datasample] += 1
            else:
                counters_dict[datasample] = 1
    sampled_data_epochtimes = [datasample for datasample in range(self. →
↪ to_datasample_epochtime(to_epochtime(self.starting_datetime)), self. →
↪ to_datasample_epochtime(to_epochtime(self.ending_datetime)), self. →
↪ seconds_in_timequantum)]
    sampled_data_values = []
    for i in range(len(sampled_data_epochtimes)):
        if sampled_data_epochtimes[i] in counters_dict:
            sampled_data_values.append(counters_dict[sampled_data_epochtimes[i]])
        else:
            sampled_data_values.append(0)
    return sampled_data_epochtimes, sampled_data_values


def get_sampled_measurement(self, variable_name: str, remove_outliers: bool=True, →
↪ outlier_sensitivity: float=5):
    data_epochtimes, data_values = self.get_raw_measurement(variable_name, →
↪ remove_outliers, outlier_sensitivity)
    if len(data_epochtimes) == 0:
        return None
    augmented_data_epochtimes = [self.to_datasample_epochtime(to_epochtime(self. →
↪ starting_datetime))]
    augmented_data_values = [data_values[0]]
    for i in range(len(data_epochtimes)):
        data_epochtime = data_epochtimes[i]
        for epochtime in range(self.to_datasample_epochtime(augmented_data_epochtimes →
↪ [-1]) + self.seconds_in_timequantum, self.to_datasample_epochtime(data_epochtime), →
↪ self.seconds_in_timequantum):
            augmented_data_epochtimes.append(epochtime)
            augmented_data_values.append(augmented_data_values[-1])
        if self.to_datasample_epochtime(data_epochtime) > self.to_datasample_epochtime( →
↪ augmented_data_epochtimes[-1]):
            augmented_data_epochtimes.append(self.to_datasample_epochtime(data_epochtime) →
↪ )
            augmented_data_values.append(augmented_data_values[-1])
        augmented_data_epochtimes.append(data_epochtime)
        augmented_data_values.append(data_values[i])
    for epochtime in range(self.to_datasample_epochtime(augmented_data_epochtimes[-1]), →
↪ to_epochtime(self.ending_datetime), self.seconds_in_timequantum):
        augmented_data_epochtimes.append(epochtime + self.seconds_in_timequantum)
        augmented_data_values.append(augmented_data_values[-1])
    sampled_data_epochtimes, sampled_data_values = [], []
    integrator = 0
    for i in range(1,len(augmented_data_epochtimes)):
```

```
            previous_epochtime = augmented_data_epochtimes[i − 1]
            previous_datasample_epochtime = self.to_datasample_epochtime( →
    ↪ augmented_data_epochtimes[i−1])
            epochtime = augmented_data_epochtimes[i]
            datasample_epochtime = self.to_datasample_epochtime(augmented_data_epochtimes[i →
    ↪ ])
            previous_value = augmented_data_values[i − 1]
            integrator += (epochtime − previous_epochtime) ∗ previous_value
            if datasample_epochtime > previous_datasample_epochtime:
                sampled_data_epochtimes.append(previous_datasample_epochtime)
                sampled_data_values.append(integrator / self.seconds_in_timequantum)
                integrator = 0
        return sampled_data_epochtimes, sampled_data_values

    def to_datasample_epochtime(self, epochtime):
        return (epochtime // self.seconds_in_timequantum) ∗ self.seconds_in_timequantum

    def close(self):
        self.connection.commit()
        self.connection.close()


def discretize(value, bounds=((0,0), (0,1), (1,1))):
    for i in range(len(bounds)):
        if value >= bounds[i][0] and value<= bounds[i][1]:
            return i
    return −1


def average_error(actual_occupants):
    level_midpoint=[]
    i = 0
    for i in range(0,len(actual_occupants),1):
        x = actual_occupants[i]
        if (x == 0 ):
            level_midpoint.append(0)
        else:
            if (0<actual_occupants[i]<2):
                level_midpoint.append(1)
            else:
                if (actual_occupants[i]>=2):
                    level_midpoint.append(3.15)
    return level_midpoint

def decision_tree(label, feature1, feature2, feature3, test_label, test_feature1, test_feature2, →
    ↪ test_feature3):
    X, Y = [], []
    for i in range(0,len(label)):
        training_features = [feature1[i],feature2[i],feature3[i]]
        X.append(training_features)
        Y.append(label[i])

    classifier = tree.DecisionTreeClassifier(random_state=0,criterion='entropy',max_depth= →
    ↪ None)
    classifier.fit(X, Y)
```

```
    features_importance = classifier.feature_importances_
    X_t = []
    for i in range(0, len(test_label)):
        test_features = [test_feature1[i],test_feature2[i],test_feature3[i]]
        X_t.append(test_features)
    Y_t=classifier.predict(X_t)
    accuracy = classifier.score(X_t, test_label)
    #save the tree
    with open("tree.dot", 'w') as file:
        file= tree.export_graphviz(classifier, out_file=file)
    return accuracy, features_importance, Y_t, classifier


seconds_in_sample = 1800
database_extraction = DatabaseExtraction('office4.db', '4/05/2015 00:30:00', →
    ↪ '15/05/2015 00:30:00', seconds_in_sample)
times, occupancies = database_extraction.get_sampled_measurement('labels')
times, co2in = database_extraction.get_sampled_measurement('co2')
for i in range(len(co2in)):  # removal of outliers
  if co2in[i] >= 2500:
     co2in[i] = 390
times, Tin = database_extraction.get_sampled_measurement('Temperature')
times, door = database_extraction.get_sampled_measurement('Door_contact')
times, RMS = database_extraction.get_sampled_measurement('RMS')

database_extraction = DatabaseExtraction('testing.db', '17/05/2015 00:30:00', →
    ↪ '21/05/2015 00:30:00', seconds_in_sample)
times_test, occupancies_test = database_extraction.get_sampled_measurement('labels')

time_test, co2in_test = database_extraction.get_sampled_measurement('co2')
times_test, Tin_test = database_extraction.get_sampled_measurement('Temperature')
times_test, door_test = database_extraction.get_sampled_measurement('Door_contact')
times_test, RMS_test = database_extraction.get_sampled_measurement('RMS')

label = [discretize(occupancies[i], ( (0, 2), (2, 6) )) for i in range(len(occupancies))]
label_test=[discretize(occupancies_test[i], ( (0, 2), (2, 6) )) for i in range(len(occupancies_test)) →
    ↪ ]
print("applying the decision tree classification.............")
accuracy, features_importance, estimated_occupancy, clasifier = decision_tree (label,RMS, →
    ↪ co2in, door, label_test, RMS_test, co2in_test, door_test)
labelerror=average_error(estimated_occupancy)

print("Accuracy = ",accuracy)
print("Features Importance(microphone,physical model,door →
    ↪ position) = ",features_importance)
print(metrics.classification_report(label_test, estimated_occupancy,target_names = ['class →
    ↪  0', 'class 1', 'class 1']))
error = []
for i in range(len(label_test)):
    s = labelerror[i] − occupancies_test[i]
    error.append(abs(s))
average_error = (numpy.sum(error)) / len(labelerror)
print("average_error = ", average_error)
plot(label_test, 'bo-', label='actual_level')
plot(estimated_occupancy, 'ro-', label='predicited _level')
```

```
plt.ylabel('Occupancy Level')
plt.xlabel('Time Quantum')
axis('tight')
grid()
plt.legend(('measured','estimated'), loc =0)
plt.show()
, features_importance, Y_t, clf,confusionMatrics)
```

3.7 Code for estimating occupancy using random forest

```
__author__ = 'amayrim'
__all__ = []
import sqlite3
import numpy
from statistics import stdev
from time import strftime, localtime, strptime, mktime
from matplotlib.pylab import *
from sklearn.ensemble import RandomForestClassifier


def to_epochtime(datetime):  # epoch time in seconds
    epochdate=mktime(strptime(datetime, '%d/%m/%Y %H:%M:%S'))
    return int(epochdate)


def to_datetime(epochtime):  # epoch time in seconds
    return strftime('%d/%m/%Y %H:%M:%S', localtime(int(round(epochtime, 0))))


def to_timequantum(epochtime, seconds_in_timequantum: int):# epoch time in seconds
    return (epochtime // seconds_in_timequantum) * seconds_in_timequantum


class DatabaseExtraction():

    def __init__(self, database_name: str, starting_datetime: '%d/%m/%Y %H:%M:%S', →
        ↪ ending_datetime: '%d/%m/%Y %H:%M:%S', seconds_in_sample: int=1800):
        self.connection = sqlite3.connect(database_name)
        self.starting_datetime = starting_datetime
        self.ending_datetime = ending_datetime
        self.seconds_in_timequantum = seconds_in_sample

    def get_seconds_in_sample(self):
        return self.seconds_in_timequantum

    def get_raw_measurement(self, variable_name: str, remove_outliers: bool=True, →
        ↪ outlier_sensitivity: float=5):
        cursor = self.connection.cursor()
        cursor.execute('SELECT time, val FROM %s WHERE time>=%i AND time →
        ↪ <%i ORDER BY time ASC' % (variable_name, to_epochtime(self. →
        ↪ starting_datetime) * 1000, to_epochtime(self.ending_datetime) * 1000))
        epochtimes, values = [], []
```

```
    for record in cursor.fetchall():
        epochtimes.append(int(record[0]) // 1000)
        values.append(record[1])
    if remove_outliers:
        delta_values = [values[k + 1] − values[k] for k in range(len(values)−1)]
        mean_delta = mean(delta_values)
        sigma_delta = stdev(delta_values)
        for k in range(1,len(values)−1):
            pdiff = values[k] − values[k−1]
            fdiff = values[k+1] − values[k]
            if (abs(pdiff) > mean_delta + outlier_sensitivity ∗ sigma_delta) and (abs(fdiff) > →
    ↪ mean_delta + outlier_sensitivity ∗ sigma_delta) and pdiff ∗ fdiff > 0:
                print('Outlier detected in %s at time %s: %f (replaced by →
    ↪ average value)' % (variable_name, to_datetime(epochtimes[k]), values[k]))
                values[k] = (values[k−1] + values[k+1])/2
    return epochtimes, values

def get_counter_measurement(self, variable_name: str):
    data_epochtimes, data_values = self.get_raw_measurement(variable_name, remove_outliers →
    ↪ =False)
    counters_dict = dict()
    for i in range(len(data_epochtimes)):
        if data_values[i] == 1:
            datasample = self.to_datasample_epochtime(data_epochtimes[i])
            if datasample in counters_dict:
                counters_dict[datasample] += 1
            else:
                counters_dict[datasample] = 1
    sampled_data_epochtimes = [datasample for datasample in range(self. →
    ↪ to_datasample_epochtime(to_epochtime(self.starting_datetime)), self. →
    ↪ to_datasample_epochtime(to_epochtime(self.ending_datetime)), self. →
    ↪ seconds_in_timequantum)]
    sampled_data_values = []
    for i in range(len(sampled_data_epochtimes)):
        if sampled_data_epochtimes[i] in counters_dict:
            sampled_data_values.append(counters_dict[sampled_data_epochtimes[i]])
        else:
            sampled_data_values.append(0)
    return sampled_data_epochtimes, sampled_data_values


def get_sampled_measurement(self, variable_name: str, remove_outliers: bool=True, →
    ↪ outlier_sensitivity: float=5):
    data_epochtimes, data_values = self.get_raw_measurement(variable_name, remove_outliers →
    ↪ , outlier_sensitivity)
    if len(data_epochtimes) == 0:
        return None
    augmented_data_epochtimes = [self.to_datasample_epochtime(to_epochtime(self. →
    ↪ starting_datetime))]
    augmented_data_values = [data_values[0]]
    for i in range(len(data_epochtimes)):
        data_epochtime = data_epochtimes[i]
        for epochtime in range(self.to_datasample_epochtime(augmented_data_epochtimes[−1]) →
    ↪ + self.seconds_in_timequantum, self.to_datasample_epochtime(data_epochtime), self. →
    ↪ seconds_in_timequantum):
```

```
            augmented_data_epochtimes.append(epochtime)
            augmented_data_values.append(augmented_data_values[-1])
        if self.to_datasample_epochtime(data_epochtime) > self.to_datasample_epochtime( →
    ↪ augmented_data_epochtimes[-1]):
            augmented_data_epochtimes.append(self.to_datasample_epochtime(data_epochtime))
            augmented_data_values.append(augmented_data_values[-1])
        augmented_data_epochtimes.append(data_epochtime)
        augmented_data_values.append(data_values[i])
    for epochtime in range(self.to_datasample_epochtime(augmented_data_epochtimes[-1]), →
    ↪ to_epochtime(self.ending_datetime), self.seconds_in_timequantum):
        augmented_data_epochtimes.append(epochtime + self.seconds_in_timequantum)
        augmented_data_values.append(augmented_data_values[-1])
    sampled_data_epochtimes, sampled_data_values = [], []
    integrator = 0
    for i in range(1,len(augmented_data_epochtimes)):
        previous_epochtime = augmented_data_epochtimes[i - 1]
        previous_datasample_epochtime = self.to_datasample_epochtime( →
    ↪ augmented_data_epochtimes[i-1])
        epochtime = augmented_data_epochtimes[i]
        datasample_epochtime = self.to_datasample_epochtime(augmented_data_epochtimes[i])
        previous_value = augmented_data_values[i - 1]
        integrator += (epochtime - previous_epochtime) * previous_value
        if datasample_epochtime > previous_datasample_epochtime:
            sampled_data_epochtimes.append(previous_datasample_epochtime)
            sampled_data_values.append(integrator / self.seconds_in_timequantum)
            integrator = 0
    return sampled_data_epochtimes, sampled_data_values

def to_datasample_epochtime(self, epochtime):
    return (epochtime // self.seconds_in_timequantum) * self.seconds_in_timequantum

def close(self):
    self.connection.commit()
    self.connection.close()


def discretize(value, bounds=((0,0), (0,1), (1,1))):
    for i in range(len(bounds)):
        if value >= bounds[i][0] and value<= bounds[i][1]:
            return i
    return -1


def average_error(actual_occupants):
    level_midpoint=[]
    i = 0
    for i in range(0,len(actual_occupants),1):
        x = actual_occupants[i]
        if (x == 0 ):
            level_midpoint.append(0)
        else:
            if (0<actual_occupants[i]<2):
                level_midpoint.append(1)
            else:
                if (actual_occupants[i]>=2):
```

```
                level_midpoint.append(3.15)
    return level_midpoint

def decision_tree(label, feature1, feature2, feature3, test_label, test_feature1, test_feature2, →
    ↪ test_feature3):
    X, Y = [], []
    for i in range(0,len(label)):
        training_features = [feature1[i],feature2[i],feature3[i]]
        X.append(training_features)
        Y.append(label[i])

    classifier = RandomForestClassifier(n_estimators=10, random_state=0, criterion='entropy →
    ↪ ',max_depth=None)#n_estimators is the number of trees in the forest
    classifier.fit(X, Y)
    features_importance = classifier.feature_importances_
    X_t = []
    for i in range(0, len(test_label)):
        test_features = [test_feature1[i],test_feature2[i],test_feature3[i]]
        X_t.append(test_features)
    Y_t=classifier.predict(X_t)
    accuracy = classifier.score(X_t, test_label)
    #save the tree

    return accuracy, features_importance, Y_t, classifier


seconds_in_sample = 1800
database_extraction = DatabaseExtraction('office4.db', '4/05/2015 00:30:00', ' →
    ↪ 15/05/2015 00:30:00', seconds_in_sample)
times, occupancies = database_extraction.get_sampled_measurement('labels')
times, co2in = database_extraction.get_sampled_measurement('co2')
for i in range(len(co2in)):  # removal of outliers
  if co2in[i] >= 2500:
     co2in[i] = 390
times, Tin = database_extraction.get_sampled_measurement('Temperature')
times, door = database_extraction.get_sampled_measurement('Door_contact')
times, RMS = database_extraction.get_sampled_measurement('RMS')

database_extraction = DatabaseExtraction('testing.db', '17/05/2015 00:30:00', ' →
    ↪ 21/05/2015 00:30:00', seconds_in_sample)
times_test, occupancies_test = database_extraction.get_sampled_measurement('labels')

time_test, co2in_test = database_extraction.get_sampled_measurement('co2')
times_test, Tin_test = database_extraction.get_sampled_measurement('Temperature')
times_test, door_test = database_extraction.get_sampled_measurement('Door_contact')
times_test, RMS_test = database_extraction.get_sampled_measurement('RMS')

label = [discretize(occupancies[i], ( (0, 2), (2, 6) )) for i in range(len(occupancies))]
label_test=[discretize(occupancies_test[i], ( (0, 2), (2, 6) )) for i in range(len(occupancies_test))]
print("applying the decision tree classification.............")
accuracy, features_importance, estimated_occupancy, clasifier = decision_tree (label,RMS, co2in →
    ↪ , door, label_test, RMS_test, co2in_test, door_test)
labelerror=average_error(estimated_occupancy)

print("Accuracy = ",accuracy)
```

```
print("Features Importance(microphone,physical model,door position →
    ↪ ) = ",features_importance)
error = []
for i in range(len(label_test)):
   s = labelerror[i] − occupancies_test[i]
   error.append(abs(s))
average_error = (numpy.sum(error)) / len(labelerror)
print("average_error = ", average_error)
plot(label_test, 'bo-', label='actual_level')
plot(estimated_occupancy, 'ro-', label='predicited _level')
plt.ylabel('Occupancy Level')
plt.xlabel('Time Quantum')
axis('tight')
grid()
plt.legend(('measured','estimated'), loc =0)
plt.show()
```