SG003/0106: scigate.org/courses

Data-driven Computational Optimization [powered by Robustness + Python]

Instructor: Mohammad Namakshenas *Date: July 2022*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

Contents

Mohammad Namakshenas [m.n.shenas@gmail.com]

1 Data-driven decision making

$$\begin{aligned} \max \quad & x_1 + x_2 \\ \text{s.t.} \quad & 2x_1 + 3x_2 \leq 8 \\ & x_1 + 4x_2 \leq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

1.1 The Bed of Procrustes: Motivation to be Data-driven

In Greek mythology, PROCRUSTES was a son of Poseidon and had a stronghold between Athens and Eleusis. He was famous for an unusual form of 'hospitality'. He possessed an iron bed (the Bed of Procrustes) and would invite passers-by to spend the night. Those that proved 'too tall' would have any 'surplus' amputated, whilst those that were 'too short' would be 'stretched' to fit his bed.

Yes, it is OK to squeeze the world into crisp ideas, reductive categories, specific vocabularies, and prepackaged narratives, however, on the occasion, it would have disastrous consequences.

Back in 2018, one of msc students asked me to collaborate on a project. Prior to our collaboration, I invited him to briefly delve into the root of the problem and why we should do that project. He was a genioun mastery over supply chain problems especially perishable supply chains, and surprisingly, he insisted to translate the problem into a perishable supply chain since his dissertation title was approved to be something like "a new approach to perishable supply chain blo blo" by the guiding board of the university.

I advised him to be the fearless Theseus. Theseus made Procrustes lie in his own bed. Then, to make him fit in it to the customary perfection, he decapitated him.

1.2 Second Aphorism: Rumi's Elephant in the Dark

Some Hindus have an elephant to show. No one here has ever seen an elephant. They bring it at night to a dark room. One by one, we go in the dark and come out saying how we experience the animal. One of us happens to touch the trunk. A water-pipe kind of creature. Another, the ear. A very strong, always moving back and forth, fan-animal. Another, the leg. I find it still, like a column on a temple. Another touches the curved back. A leathery throne. Another the cleverest, feels the tusk. A rounded sword made of porcelain. He is proud of his description. Each of us touches one place and understands the whole that way. The palm and the fingers feeling in the dark are how the senses explore the reality of the elephant.

2 Python: An Interface for Data Organization+Analysis and Optimization

In this section, we are going to understand the python functionalities for our purpose: Optimization.

2.1 A Quick Intro to Python

Python is a dynamic, interpreted (bytecode-compiled) language. There are no type declarations of variables, parameters, functions, or methods in source code. This makes the code short and flexible, and you lose the compile-time type checking of the source code. Python tracks the types of all values at runtime and flags code that does not make sense as it runs.

Like C++ and Java, Python is case sensitive so "a" and "A" are different variables. The end of a line marks the end of a statement, so unlike C++ and Java, Python does not require a semicolon at the end of each statement. Comments begin with a '#' and extend to the end of the line. Python source files use the ".py" extension and are called "modules."

2.2 Installing Anaconda+Jupiter lab

Anaconda is an open-source distribution of the Python and R programming languages for data science that aims to simplify package management and deployment. Package versions in Anaconda are managed by the package management system, conda, which analyzes the current environment before executing an installation to avoid disrupting other frameworks and packages.

The Anaconda distribution comes with over 250 packages automatically installed. Over 7500 additional open-source packages can be installed from PyPI as well as the conda package and virtual environment manager. It also includes a GUI (graphical user interface), Anaconda Navigator, as a graphical alternative to the command line interface. Anaconda Navigator is included in the Anaconda distribution, and allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages, install them in an environment, run the packages and update them.

To install the Anaconda, go to the following link and find the best distribution that fits your OS and system architecture (32 or 64 bit):

```
https://www.anaconda.com/products/distribution
```

Figure 1: Anaconda Logo

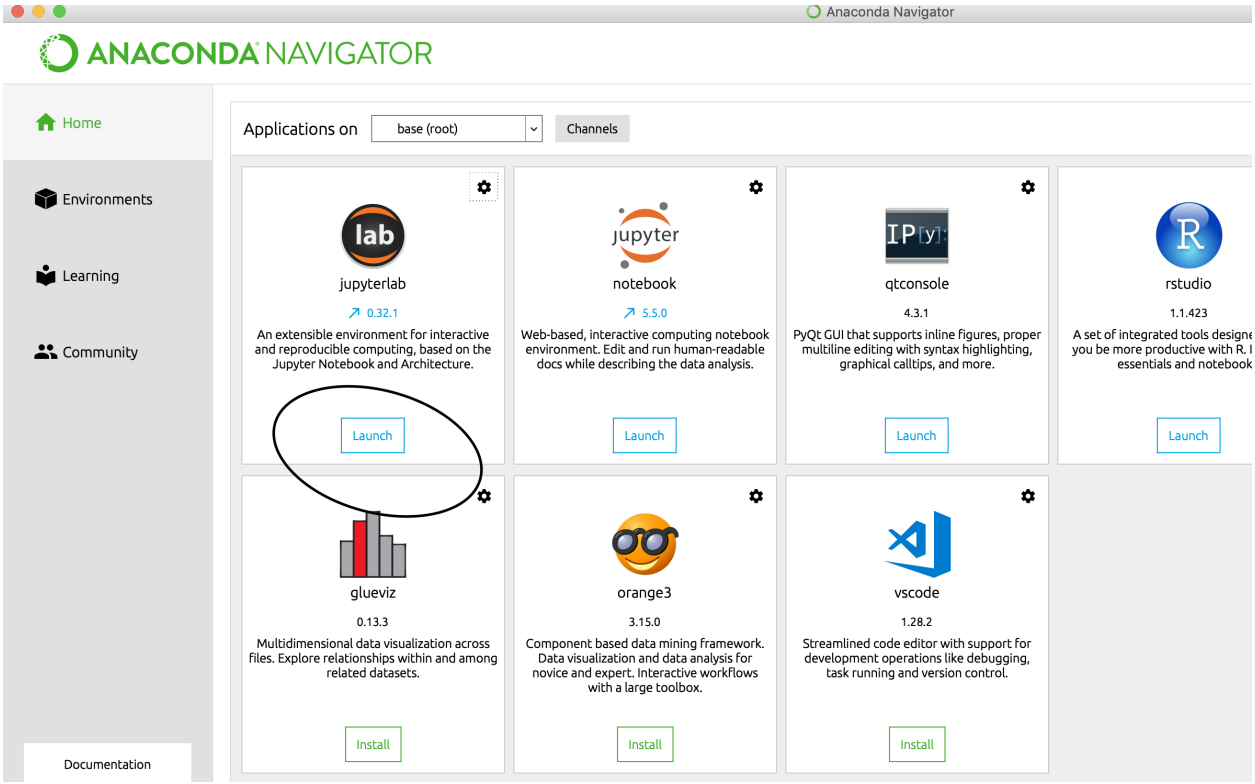

Figure 2: jupyterlab in the Anaconda Navigator

WARNING: Please install Anaconda with COMPLETELY default configuration!

After the installation, you need to check that whether the following paths are available in the Environment Variable (in case your OS is windows). If not, just add them to the system path.

```
C:\Users\YOUR_PC_NAME\anaconda3
C:\Users\YOUR_PC_NAME\anaconda3\Scripts
```

In the search bar, type ”anaconda” and click on ”anaconda prompt (anaconda3)”. This will

open a command-line shell that understands the python commands. Type the following command:

```
jupyter lab
```

So, what is jupyterlab? JupyterLab is a next-generation web-based user interface for Project Jupyter. JupyterLab enables you to work with documents and activities such as Jupyter notebooks, text editors, terminals, and custom components in a flexible, integrated, and extensible manner. JupyterLab also offers a unified model for viewing and handling data formats. Notebook cell outputs can be mirrored into their own tab, side by side with the notebook, enabling simple dashboards with interactive controls backed by a kernel. Kernel-backed documents enable code in any text file (Markdown, Python, R, LaTeX, etc.) to be run interactively in any Jupyter kernel.

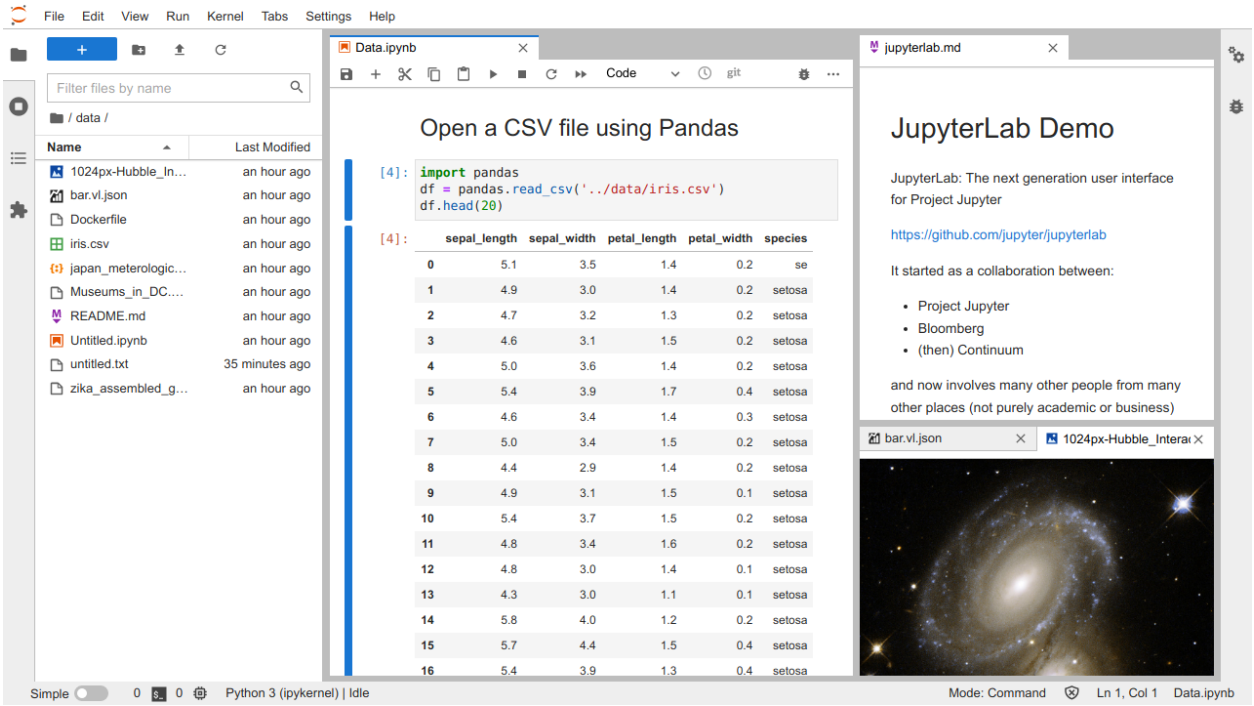

Figure 3: jupyterlab interface

Open a python3 notebook, then in the first cell type:

```
print("Hello DDRO course")
```

By holding shift and pressing enter, you can evaluate a cell. If the above code is executed normally with the printed sentence, then you're welcome!

WARNING: We are not going to cover all commands and functionalities in python. All we need to know is the functionalities and concepts to understand our data and quickly delve into optimization.

2.3 Python Data Structures

Organizing, managing and storing data is important as it enables easier access and efficient modifications. Data Structures allows you to organize your data in such a way that enables you to store collections of data, relate them and perform operations on them accordingly.

Python has implicit support for Data Structures which enable you to store and access data. These structures are called List, Dictionary, Tuple and Set.

Python allows its users to create their own Data Structures enabling them to have full control over their functionality. The most prominent Data Structures are Stack, Queue, Tree, Linked List and so on which are also available to you in other programming languages.

Remark. Python language is case-sensitive! with indented statements! and do not need semicolon at the end of any statement!

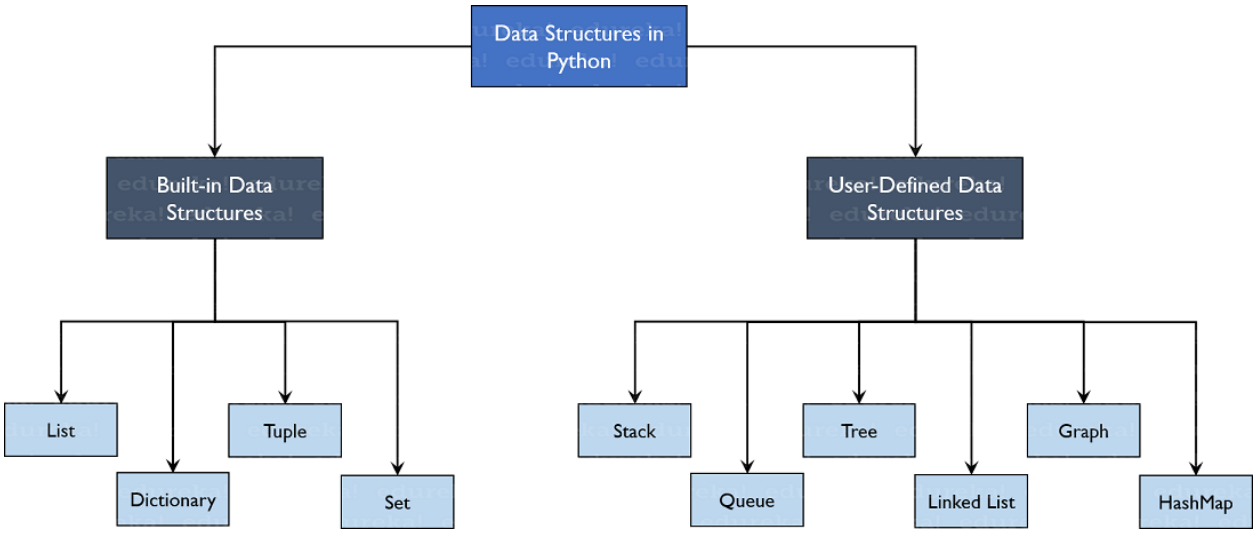

Figure 4: Types of Data Structures in Python

Lists are used to store data of different data types in a sequential manner. There are addresses assigned to every element of the list, which is called as Index. The index value starts from 0 and goes on until the last element called the positive index. There is also

negative indexing which starts from -1 enabling you to access elements from the last to first. Let us now understand lists better with the help of an example program.

To create a list, you use the square brackets and add elements into it accordingly. If you do not pass any elements inside the square brackets, you get an empty list as the output.

```
my_list = [1, 2, 3, 5, 9] #creating list with data
print(my_list)
my_list = [1, 2, 3, 'abc', 3.132, 245697] #creating list with data
print(my_list)
```

There are some useful functions to manipulate and access an already created list:

- Adding Elements
  - The `append()` function adds all the elements passed to it as a single element.
  - The `extend()` function adds the elements one-by-one into the list.
  - The `insert()` function adds the element passed to the index value and increase the size of the list too.
- Deleting Elements
  - To delete elements, use the `del` keyword which is built-in into Python but this does not return anything back to us.
  - If you want the element back, you use the `pop()` function which takes the index value.
  - To remove an element by its value, you use the `remove()` function.
- Accessing Elements
- Other Functions
  - The `len()` function returns to us the length of the list.
  - The `index()` function finds the index value of value passed where it has been encountered the first time.
  - The `count()` function finds the count of the value passed to it.
  - The `sorted()` and `sort()` functions do the same thing, that is to sort the values of the list.
  - The `sorted()` has a return type whereas the `sort()` modifies the original list.

```
### Adding Elements
my_list = [1, 2, 3]
print(my_list)
my_list.append([3, 4]) #add as a single element
print(my_list)
my_list.extend([5, 6]) #add as different elements
print(my_list)
my_list.insert(1, 7) #add element i
print(my_list)

### Deleting Elements
my_list = [1, 2, 3, 'a', 3.132, 10, 30]
del my_list[3] #delete element at index 5
print(my_list)
my_list.remove('3.132') #remove element with value
print(my_list)
a = my_list.pop(1) #pop element from list
print('Popped Element: ', a, ' List remaining: ', my_list)
my_list.clear() #empty the list
print(my_list)

### Accessing Elements
my_list = [1, 2, 3, 'a', 3.132, 10, 30]
print(my_list) #access all elements
print(my_list[3]) #access index 3 element
print(my_list[0:2]) #access elements from 0 to 1 and exclude 2
print(my_list[::-1]) #access elements in reverse

### Other Functions
my_list = [1, 2, 3, 10, 30, 10]
print(len(my_list)) #find length of list
print(my_list.index(10)) #find index of element that occurs first
print(my_list.count(10)) #find count of the element
print(sorted(my_list)) #print sorted list but not change original
my_list.sort(reverse=True) #sort original list
print(my_list)
```

Remark. Python lists are 0-indexed. So the first element is 0, second is 1, so on. So if the there are n elements in a list, the last element is n-1! Python also supports indexing from the end, that is, negative indexing. This means the last value of a sequence has an index of -1, the second last -2, and so on.

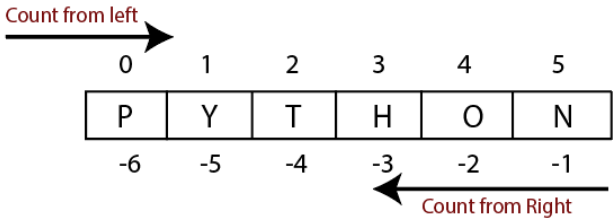

Figure 5: Python List Indexation

Dictionaries are used to store key-value pairs. To understand better, think of a phone directory where hundreds and thousands of names and their corresponding numbers have been added. Now the constant values here are Name and the Phone Numbers which are called as the keys. And the various names and phone numbers are the values that have been fed to the keys. If you access the values of the keys, you will obtain all the names and phone numbers. So that is what a key-value pair is. And in Python, this structure is stored using Dictionaries. Let us understand this better with an example program.

```
### Changing and Adding key, value pairs
my_dict = {'Tehran': 32, 'Yazd': 25, 'Arak': 15}
print(my_dict)
my_dict['Yazd'] = 28 #changing element
print(my_dict)
my_dict['Semnan'] = 19 #adding key-value pair
print(my_dict)

### Deleting key, value pairs
a = my_dict.pop('Arak') #pop element
print('Value:', a)
print('Dictionary:', my_dict)
b = my_dict.popitem() #Remove the last item from the dictionary
print('Key, value pair:', b)
print('Dictionary', my_dict)
my_dict.clear() #empty dictionary
print('n', my_dict) # n for new line

### Accessing Elements
print(my_dict['Tehran']) #access elements using keys

### Other Functions
print(my_dict.keys()) #get keys
print(my_dict.values()) #get values
print(my_dict.items()) #get key-value pairs
```

Tuples are the same as lists are with the exception that the data once entered into the tuple cannot be changed no matter what. The only exception is when the data inside the tuple is mutable, only then the tuple data can be changed. The example program will help you understand better.

```
# Creating a Tuple
my_tuple = (1, 2, 3) #create tuple
print(my_tuple)

# Accessing Elements
my_tuple = (1, 2, 3)
print(my_tuple[0])
print(my_tuple[:])

# Appending Elements
my_tuple = (1, 2, 3)
my_tuple = my_tuple + (4, 5, 6) #add elements VERY ROBUST
print(my_tuple)
```

Sets are a collection of unordered elements that are unique. Meaning that even if the data is repeated more than one time, it would be entered into the set only once. It resembles the sets that you have learned in arithmetic. The operations also are the same as is with the arithmetic sets. An example program would help you understand better.

```
# Creating a set
my_set = {1, 2, 3, 4, 5, 5, 5} #create set
print(my_set)

#Adding elements
my_set = {1, 2, 3}
my_set.add(4) #add element to set
print(my_set)

#Operations in sets
my_set = {1, 2, 3, 4}
my_set_2 = {3, 4, 5, 6}
print(my_set.union(my_set_2), '----------', my_set | my_set_2)
print(my_set.intersection(my_set_2), '----------', my_set & my_set_2)
print(my_set.difference(my_set_2), '----------', my_set - my_set_2)
```

NumPy Arrays: NumPy stands for Numerical Python. It is a Python library used for working with an array. In Python, we use the list for purpose of the array but it's slow to process. NumPy array is a powerful N-dimensional array object and its use in linear algebra,

Fourier transform, and random number capabilities. It provides an array object much faster than traditional Python lists.

Remark. The most important difference between list and array is that the Numpy will automatically assign a type for the whole array. e.g., float64, integer.

```
# importing numpy module
import numpy as np

# creating list
my_list = [1, 2, 3, 4]

# creating numpy array
my_array = np.array(my_list)

# See the difference
print("List in python : ", my_list)
print("Numpy Array in python :", my_array)
print(type(my_list))
print(type(my_array))

### creating 2-dimensional array

# creating list
list_1 = [1, 2, 3, 4]
list_2 = [5, 6, 7, 8]
list_3 = [9, 10, 11, 12]

# creating numpy array
sample_array = np.array([list_1,list_2,list_3])

print(sample_array)

### the most important difference between list and array

# Creating the array
sample_array_1 = np.array([[0, 4, 2]])

sample_array_2 = np.array([0.2, 0.4, 2.4])

# display data type
print("Data type of the array 1 :",
sample_array_1.dtype)
```

```
print("Data type of array 2 :",
sample_array_2.dtype)
```

2.4 Handling Loops and Logics

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

```
# Structure of a loop
# Don't forget indentation.
for val in sequence:
        loop body
```

range function is a sequence builder. We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers). We can also define the start, stop and step size as range(start, stop,step_size). step_size defaults to 1 if not provided.

```
# range(start, stop,step_size)
print(list(range(2, 20, 3)))

#iterate over the list (or sequence) using index
for i in range(10):
        print("the number is", i)
```

Decision making is required when we want to execute a code only if a certain condition is satisfied. The **if. . . elif. . . else** statement is used in Python for decision making.

```
# basic structure of if-statement
if test expression:
        statement(s)
```

```
'''In this program,
we check if the number is positive or
negative or zero and
display an appropriate message'''
```

```
num = 3.4

if num > 0:
        print("Positive number")
elif num == 0:
        print("Zero")
else:
        print("Negative number")
```

We can combine for and if statements:

```
# show the demand of specific city

city_name = 'kerman'

city_distances = {'tehran': 90, 'esfahan': 55, 'kerman': 77}

for c in city_distances:
        if city_name == c:
                print(city_distances[c])
                break
        else:
                print('No entry with that name found.')
```

Remark. Four ways to import libraries:

```
#1
from math import pi

#2
from math import pi, sqrt

#3
import math as m
import numpy as np
import pandas as pd

#4
from math import sqrt as sq
```

2.5 Array Programming with NumPy

Data manipulation in Python is nearly synonymous with NumPy array manipulation. This section will present several examples of using NumPy array manipulation to access data and subarrays, and to split, reshape, and join the arrays. While the types of operations shown here may seem a bit dry and pedantic, they comprise the building blocks of many other examples used throughout the course.

We'll cover a few categories of basic array manipulations here:

- *Attributes of arrays*: Determining the size, shape, memory consumption, and data types of arrays
- *Indexing of arrays*: Getting and setting the value of individual array elements
- *Slicing of arrays*: Getting and setting smaller subarrays within a larger array
- *Reshaping of arrays*: Changing the shape of a given array
- *Joining and splitting of arrays*: Combining multiple arrays into one, and splitting one array into many

NumPy Array Attributes

```
import numpy as np
x = np.array([[1,2,3,4],
                          [5,6,7,8]
                          [9,10,11,12]])

print("x ndim: ", x.ndim)
print("x shape:", x.shape)
print("x size: ", x.size)

### output
# x ndim:  3
# x shape: (3, 4)
# x size: 12

# number of columns
x.shape[1]
```

Array Indexing: Accessing Single Elements

```
import numpy as np
y = np.array([5,4,3,2,1])

y[0]

y[3]

y[-1]

x = np.array([[1,2,3,4],
[5,6,7,8]
[9,10,11,12]])

x[2, 0]

# can also be modified
x2[0, 0] = 12

###NumPy arrays have a fixed type!
y[0] = 3.14 # this will be truncated!
print(y.dtype)

#changing the type
y_p = y.astype(np.float64)
y_p[0] = 3.14
print(y_p)
```

Remark. NumPy arrays have a fixed type!

numpy.float32: "python float"
numpy.float64: "python float"
numpy.uint32: "python int"
numpy.int16: "python int"

Array Slicing: Accessing Subarrays

Just as we can use square brackets to access individual array elements, we can also use them to access subarrays with the slice notation, marked by the colon (:) character. The NumPy slicing syntax follows that of the standard Python list; to access a slice of an array x, use this:

```
### general syntax
```

```
x[start:stop:step]
```

```
### creating a linearly increasing array
# arange(start,stop,step)
x = np.arange(10)
print(x)

# slicing 1d arrays
x[5:]  # elements after index 5
x[4:7]  # middle sub-array
x[::2]  # even elements
x[1::2]  # even elements, starting at index 1
x[::-1]  # reversing the array
x[5::-2]  # reversed every other from index 5

# slicing 2d arrays
y = np.array([[1,2,3,4],
[5,6,7,8]
[9,10,11,12]])
y[:2, :3]  # two rows, three columns
y[:, 0] # first column
y[0, :]  # first row
```

Reshaping of Arrays

```
# quickly creating a 3,4 matrix (2d array) using reshape
x = np.arange(12).reshape((3,4))

# transposing
np.transpose(x)
```

Initializing a Numpy Array

```
#directly from list
x = [1,2,3]
np.array(x)

#array of ones
np.ones((5,2,3))
```

```
#array of zeros
np.zeros(5)

#array of any value
np.full((2,3),10)
#output
#array([[10, 10, 10],
#                [10, 10, 10]])

#sequential or evenly spaced values
np.arange(10)

# I want to produce an array of size 5 in which values
#should be between 2 and 3 and the step between values
#should be equal and the endpoint, i.e., 3, should not be considered.
np.linspace(2, 3, num=5, endpoint=False)

#array (3,2) of random numbers between 0 and 1
np.random.rand(3,2)

#array (3,2) of random integer numbers between 10 and 20
np.random.randint(10,20,(3,2))
```

Concatenation of Arrays

Concatenation, or joining of two arrays in NumPy, is primarily accomplished using the routines `np.concatenate`, `np.vstack`, and `np.hstack`.

```
x = np.array([1, 2, 3])
y = np.array([3, 2, 1])
np.concatenate([x, y])

# concatinate more than two arrays
z = [5, 6, 7]
print(np.concatenate([x, y, z]))

#concatinate 2d arrays
p = np.array([[1, 2, 3],
                [4, 5, 6]])
# concatenate along the first axis
```

```
np.concatenate([p, p])

# concatenate along the second axis (zero-indexed)
np.concatenate([p, p], axis=1)

# vertically stack the arrays
x = np.array([1, 2, 3])
y = np.array([[9, 8, 7],
        [6, 5, 4]])
np.vstack([x, y])

# horizontally stack the arrays
z = np.array([[99],
[99]])
np.hstack([y, z])
```

Algebraic Applications

```
# computing norm function
from numpy import linalg as LA
# || x ||_2 = sqrt(1^2 + 2^2 + 3^2 + 4^2)
x = [1,2,3,4,5]
LA.norm(x,2)

######################
# inner product (component-wise product) vs dot-product
x = [1,2,3,4,5]
y = 3
print(x*y)

x = np.array(x)
print(x*y)

#error
x = [1,2,3,4,5]
y = [6,7,8,9,10]
print(x*y) #error

# inner-product
y = [6,7,8,9,10]
```

```
x = np.array(x)
y = np.array(y)
print(x*y)

# dot-product
np.dot(x,y)

# and matrix dimensions should be consistent!
x = np.full((3,2),2)
y = np.full((3,4),5)
print(x, '\n', y)
print(np.dot(x,y)) # error
#the following is OK
print(np.dot(x.T,y))
```

Remark. In the case of a rank 1 array, the .T and .transpose() don't do anything—they both return the array.

Remark. Most of our works in this course will be based on Numpy array functionalities!

Remark. Numpy functionalities is far beyond the material presented in this section!

2.6 Serving DataFrames with Pandas

Pandas is a newer package built on top of NumPy, and provides an efficient implementation of a DataFrame. DataFrames are essentially multidimensional arrays with attached row and column labels, and often with heterogeneous types and/or missing data. As well as offering a convenient storage interface for labeled data, Pandas implements a number of powerful data operations familiar to users of both database frameworks and spreadsheet programs.

we will import Pandas under the alias pd:

```
import pandas as pd
```

In this course, we are not going to cover all pandas syntax; instead, we will cover how one can create, read, and write spreadsheets.

The idea is that we read from a compressed database, then manipulate it by Numpy functions, and finally save the results in another .csv file.

```
import numpy as np
import pandas as pd

# create random demands for five cities demanding four commodities
demand_data = np.random.randint(30,60,(5,4))

# create dataframe
city_names = ['tehran','esfahan','tabriz','qom','arak']
comodity_name = ['A','B','C','D']
df = pd.DataFrame(demand_data,index=city_names,columns=comodity_name)

#visualize dataframe
df.head()
df.tail()

#save it to csv file
df.to_csv('demand-data.csv',index=False)

#load to a dataframe
df_loaded = pd.read_csv('demand-data.csv')
df_loaded.index = city_names

#transforming dataframe to numpy array
new_array = df_loaded.to_numpy()
binary_array = np.zeros(new_array.shape)

# create a binary matrix from truncating demands
#more than spesific threshold
demand_threshold = 40
# ndindex is like range function but in nd dimensions!
for s in np.ndindex(new_array.shape):
        if new_array[s] >= demand_threshold:
                binary_array[s] = 1
print(binary_array)

#create a dataframe from the binary array and save it to a csv file
df2 = pd.DataFrame(binary_array,index=city_names,columns=comodity_name)
df2.to_csv("binary_mat.csv",index=False)
```

2.7 Visualizing Results with MatPlotLib

One of Matplotlib's most important features is its ability to play well with many operating systems and graphics backends. Matplotlib supports dozens of backends and output types, which means you can count on it to work regardless of which operating system you are using or which output format you wish. This cross-platform, everything-to-everyone approach has been one of the great strengths of Matplotlib. It has led to a large user base, which in turn has led to an active developer base and Matplotlib's powerful tools and ubiquity within the scientific Python world.

There are also other libraries like Plotly, Seaborn, GGplot, Altair, and Bokeh.

```
import matplotlib.pyplot as plt
```

We can create plots in different ways in via matplotlib:

```
### object-oriented using matplotlib
fig, ax = plt.subplots(2)

# Call plot() method on the appropriate object
ax[0].plot(x, np.sin(x))
ax[1].plot(x, np.cos(x));



### object-oriented using matplotlib (plus complete charactristic of the plot)
ax = plt.axes()
ax.plot(x, np.sin(x))
ax.set(xlim=(0, 10), ylim=(-2, 2),
xlabel='x', ylabel='sin(x)',
title='A Simple Plot');
```

We can also change the size of the canvas:

```
#on ax
ax.figure.set_size_inches(14, 10)

#or on fig
fig.set_size_inches(18.5, 10.5)
```

Another exmpale: Generate 1000 samples from normal distribution and then plot it by histogram!

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

fig = plt.figure()

data = np.random.randn(1000)

plt.hist(data)
```

Also, box plot is one of great tools to show different aspects of data in one single frame!

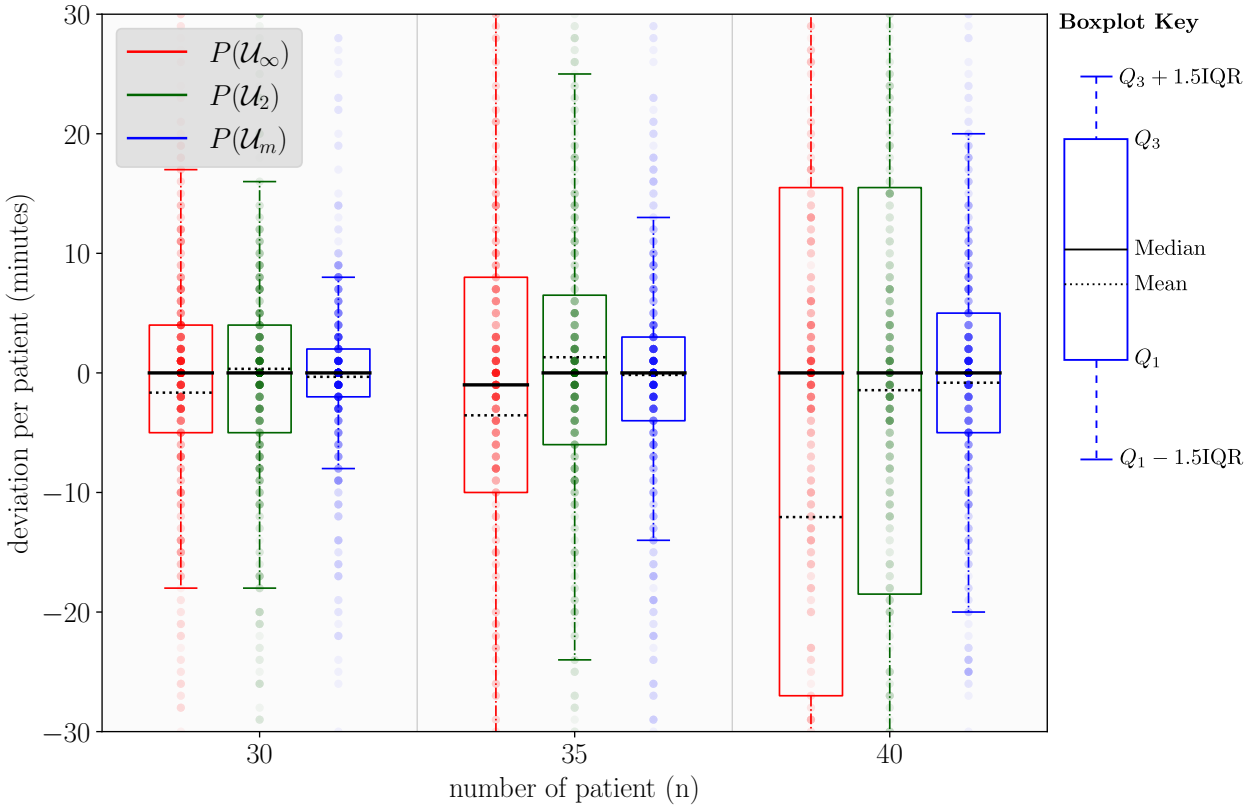

Figure 6: A sample box plot

2.8 Object-oriented Programming in Python

Motivation 1. Why do even we need to know the OOP? The classical solvers were based on column-wise programming or matrix-oriented inputs. However, in the next-gen solvers, we are able to define an instance of a problem as object via solver classes and instantiation, and add properties of that instance gradually through methods. These properties can be constraints, decision variables, and objective functions.

Motivation 2. Sometimes we want to separate an abstract layer of program form the application [or object or instantiation] side. According to Garry and Johnson (ref...), the abstract layer corresponds to "Problem" and the instantiation refers corresponds to "Instance." For example,

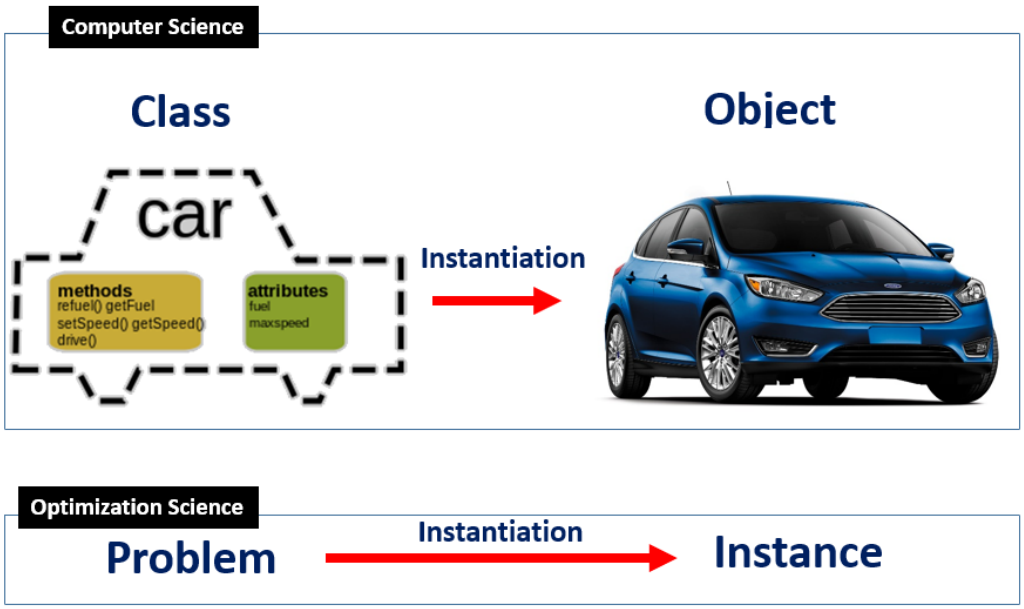

Figure 7: An example of object-oriented programming

The `__init__` special method, also known as a Constructor, is used to initialize the Book class with attributes such as title, quantity, author, and price. Moreover, we can use a specific method called `__repr__` to print our defined info about the class.

```
import numpy as np
class SupplyChain:
        def __init__(self, _supply_chain_title, _num_supplier,\
        _num_producer, _num_distribution, _num_demand_nodes):
                self.supply_chain_title = _supply_chain_title
```

```
                self.num_supplier = _num_supplier
                self.num_producer = _num_producer
                self.num_distribution = _num_distribution
                self.num_demand_nodes = _num_demand_nodes

        def get_level(self):
                return np.sign(self.num_supplier) + np.sign(self.num_producer) + \
                np.sign(self.num_distribution) + np.sign(self.num_demand_nodes) - 1

        def __repr__(self):
                return (f"A {self.get_level()} level supply chain named {self.supply_chain_title}"
                f" with {self.num_supplier} suppliers, {self.num_producer} producers,"
                f" {self.num_distribution} distributions,"
                f" and {self.num_demand_nodes} demand nodes is created.")
```

Remark. User-defined classes are named in Camel or Snake case, with the first letter capitalized. And note that how we breaked the long lines!

Then, we can create different instances of the problem.

```
# instantiating different objects from the class
# or initinializing different instances of the supply chain problem

scm1 = SupplyChain("Green Supply chain",2,3,4,5)
print(scm1)

scm2 = SupplyChain("Green Supply chain",0,3,4,5)
print(scm2)

scm3 = SupplyChain("Green Supply chain",0,3,4,0)
print(scm3)
```

Or, we can access a method of the class like `get_level()`:

```
scm2.get_level()
```

Of course we didn't cover many concepts in Python. If you want to know more functionalities in Python, a lot to go but you can study the following book which part of the Python tutorial in this course is from this book:

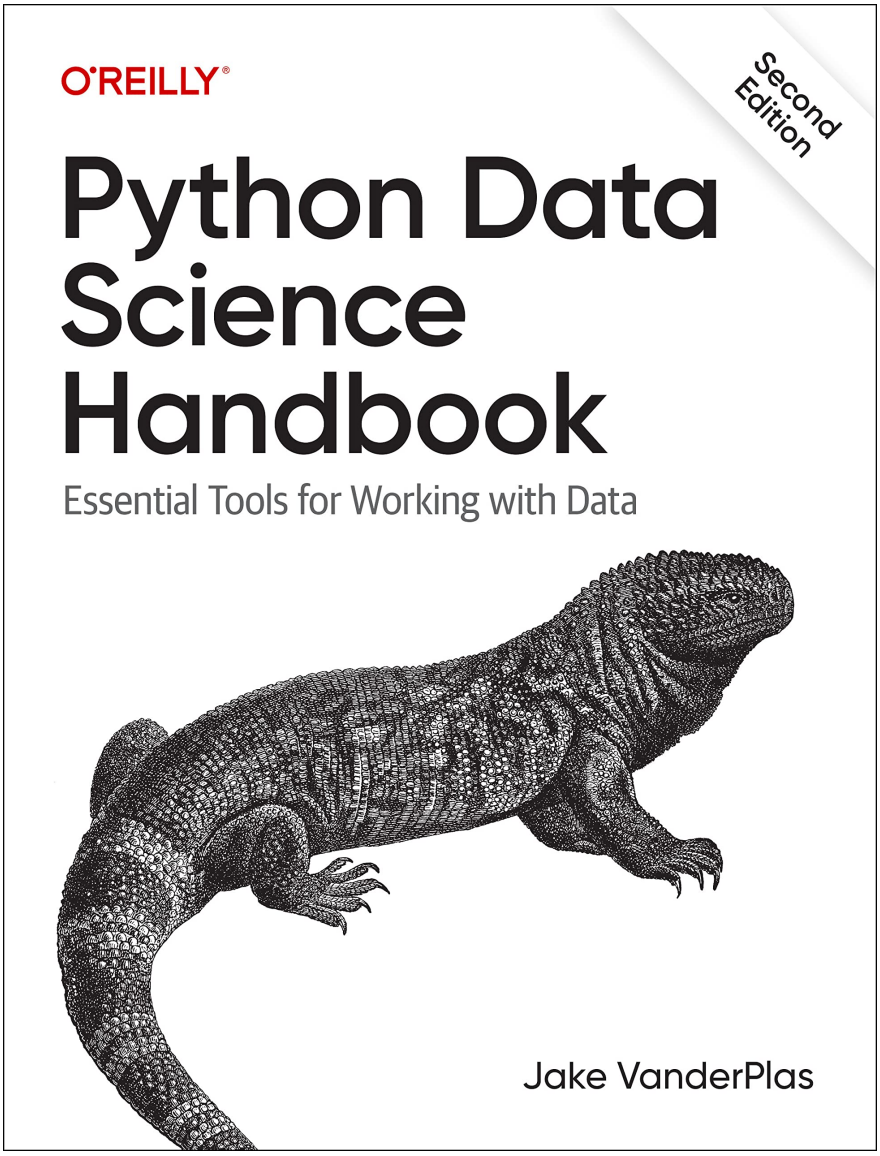

Figure 8: Book Introduction: Python Data Science Handbook

3 Computational Optimization

3.1 Optimization Classes: Convex and Non-convex

Let's divide the optimization classes into three categories:

- Convex
- Non-convex Continuous
- Non-convex Integer

We first need to define the formal definition of convexity. A convex optimization problem is one of the form

$$\begin{aligned} \min_{x} \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leq b_i, \quad \forall i \in \{1, \ldots, m\}, \end{aligned} \tag{1}$$

where functions $f_0, f_1, \ldots, g_m : \mathbb{R}^n \to \mathbb{R}$ are convex and satisfy

$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y), \tag{2}$$

for all $x, y \in \mathbb{R}^n$ and all $\alpha, \beta \in \mathbb{R}$ with $\alpha + \beta = 1$, $\alpha \geq 0$, and $\beta \geq 0$.

But, let's define our practical version of convexity:

Definition 1. The "Hessian matrix" of a multi-variable function organizes all second partial derivatives into a matrix:

$$\boldsymbol{H_f} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_n} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n x_1} & \frac{\partial^2 f}{\partial x_n x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Theorem 3.1. *Let H be a symmetric matrix. We call H a positive semi-definite (PSD) matrix if one the conditions below holds:*

- *All eigenvalues of H satisfy $\lambda_j \geq 0$.*
- *All NW (upper left) minors of H are positive.*

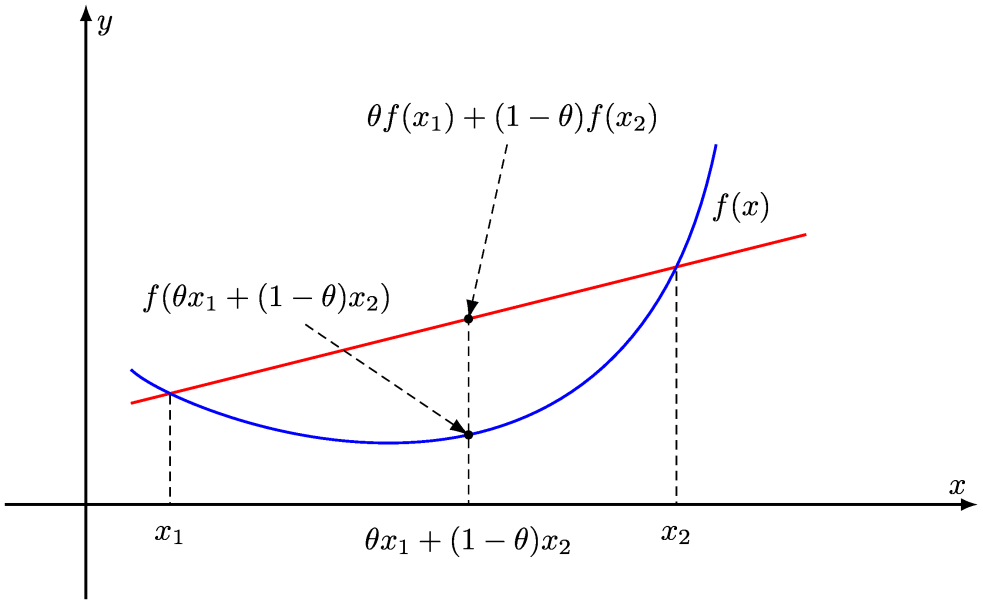

Figure 9: An illustration of a convex function

Theorem 3.2 (necessary and sufficient condition)**.** *A twice continuously differentiable function is convex if and only if its Hessian is a PSD matrix.*

Convex Optimization. Convex optimization is a subfield of mathematical optimization that studies the problem of minimizing convex functions over convex sets (or, equivalently, maximizing concave functions over convex sets). Many classes of convex optimization problems admit polynomial-time algorithms. The following are useful properties of convex optimization problems:

1. Every local minimum is a global minimum.
2. If the objective function is strictly convex, then the problem has at most one optimal point.

Some of useful convex optimization classes are as follows. 1) Linear Programming (LP) 2) Quadratic Programming (QP) subjected to convex feasible region and Positive Semi-definiteness of objective function, 3) Least Square Programming (LSP) 4) Second-order Conic Programming (SOCP), and 5) Semi-definite Programming (SDP).

Example 1. Consider the following QP problem:

$$\begin{aligned} \min \quad & 6(x_1-10)^2+4(x_2-12)^2 \qquad \boldsymbol{H_0}=\begin{bmatrix}12 & 0\\ 0 & 8\end{bmatrix} \\ \text{s.t.} \quad & x_1^2+3x_2^2\le 200, \qquad \boldsymbol{H_1}=\begin{bmatrix}2 & 0\\ 0 & 6\end{bmatrix} \\ & (x_1-6)^2+x_2^2\le 37, \qquad \boldsymbol{H_2}=\begin{bmatrix}2 & 0\\ 0 & 2\end{bmatrix} \\ & x_1,x_2\ge 0 \end{aligned}$$

For minimization problems, the objective should be convex or have a PSD Hessian like $\boldsymbol{H_0}$ and for maximization problems, it should be concave. On the other hand, the intersection of the first and the second constraint is also convex since they are both independently convex. Hence, this is a convex optimization problem.

Non-convex Continuous Class. A non-convex optimization problem is any problem where the objective or any of the constraints are non-convex, as pictured below. A non-convex function "curves up and down" – it is neither convex nor concave. A familiar example is the sine function. One of the useful methods to solve these classes is "Stochastic Gradient Descent."

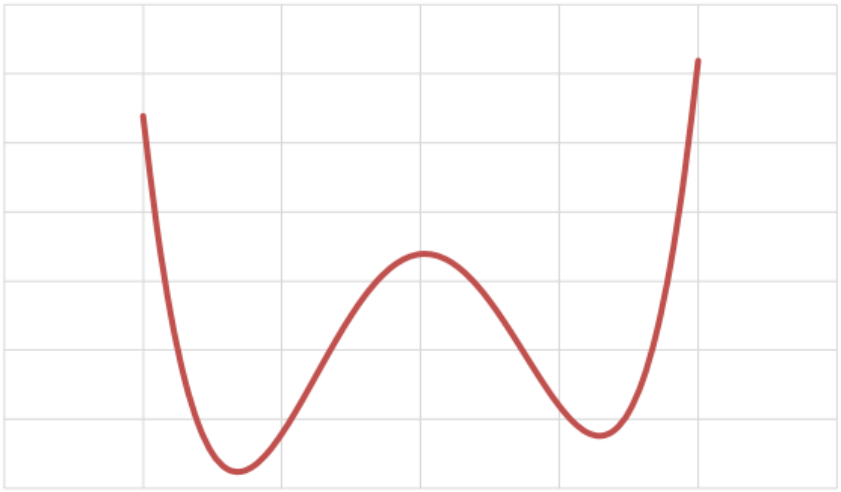

Figure 10: A non-convex feasible region

Example 2 (convertible non-convex)**.**

$$\begin{aligned} \min \quad & x+\sqrt{x} \\ \text{s.t.} \quad & x\in \text{DOM} \end{aligned}$$

However, this program can be reformulated into a convex optimization program via epigraphs ($t = \sqrt{x}$):

$$\begin{aligned} \min \quad & x + t \\ \text{s.t.} \quad & x \leq t^2, \qquad \boldsymbol{H_0} = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix} \\ & x \in \text{DOM} \end{aligned}$$

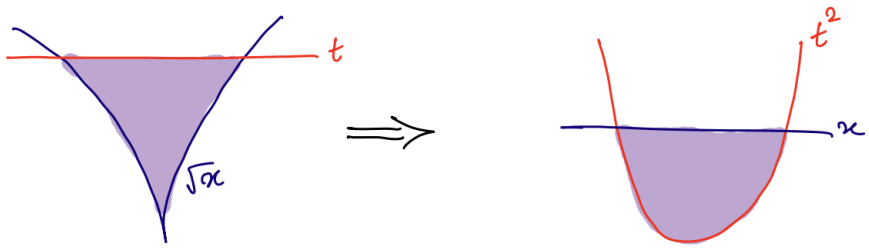

Figure 11: How to reformulate a non-convex program into a convex program

Example 3 (non-convertible non-convex: subset-sum minimization)**.**

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} x_i^2 (1 - x_i)^2 \\ \text{s.t.} \quad & x \in \text{DOM} \end{aligned}$$

Integer Non-convex Class (tautology!). Optimizing any function over a discrete space or a lattice can be formulated as an integer programming problem (Fig 12). Useful subclasses in integer optimization class are: Mixed-integer Programming, Binary Programming, Mixed-integer Quadratic Programming, Mixed-integer Conic Programming. This class is the hardest class from the point of computational complexity. Best known methods [algorithms] to solve this class are Branch and Cut and Benders Decomposition.

Some of famous integer optimization problems:

- Knapsack Problem (NP-hard in ordinary sense)
- Traveling Salesman Problem (NP-hard in strong sense or equivalently its decision problem is Co-NP-Complete)

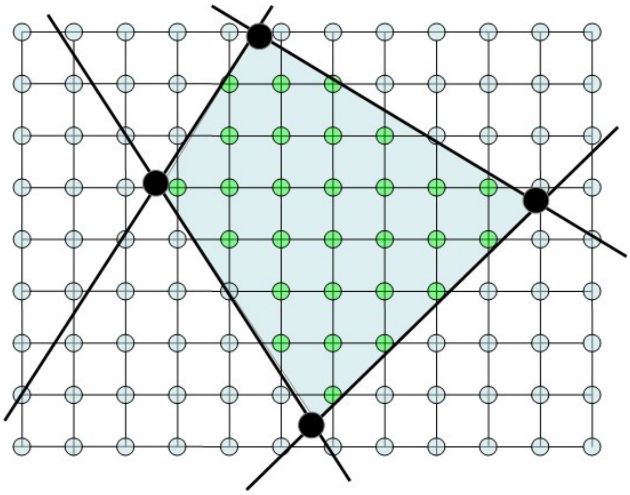

Figure 12: Integer optimization over lattice

3.2 Object-oriented Computational Optimization

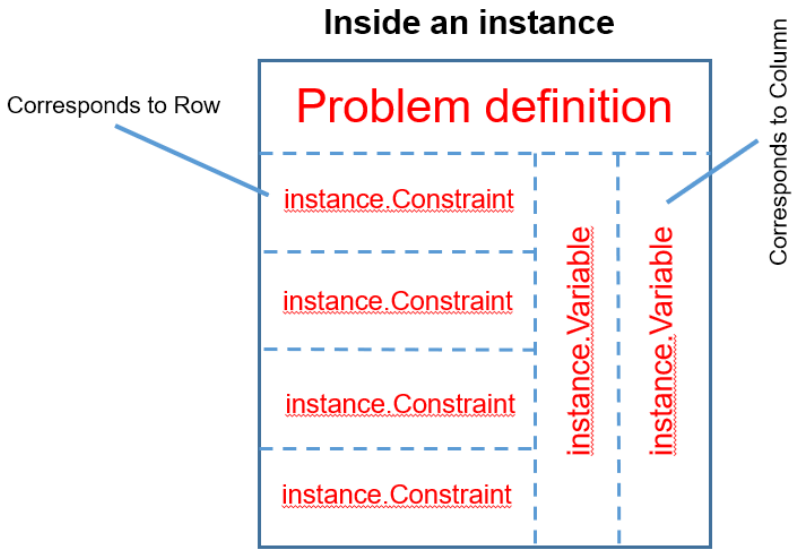

Figure 13: How changes occur inside an instance

Abstract model (problem) Initialized model (instance)

3.3 Wrappers and Optimization Engines

An optimization solver or an optimization engine (framework) is:

- a library/wrapper/package (e.g., pygurobi)
- often written in a mid-level (low-level) [high-performance] programming language like C or C++
- that incorporates one or more algorithms (e.g., barrier, simplex)
- for finding solutions to one or more classes of problem (e.g., lp).

Some of useful [only] solvers:

- Gurobi
- Mosek
- Barron
- CVX

Some of useful softwares:

- IBM Cplex
- AIMMS
- MiniZinc
- AMPL

Next, we are going to introduce gurobipy.

3.4 GurobiPy Wrapper: A Powerful Optimization Engine

This section documents the Gurobi Python interface. It begins with an overview of the global functions, which can be called without referencing any Python objects. For getting a complete overview of methods and function, just goooooooogle:

```
gurobi Python API Overview
```

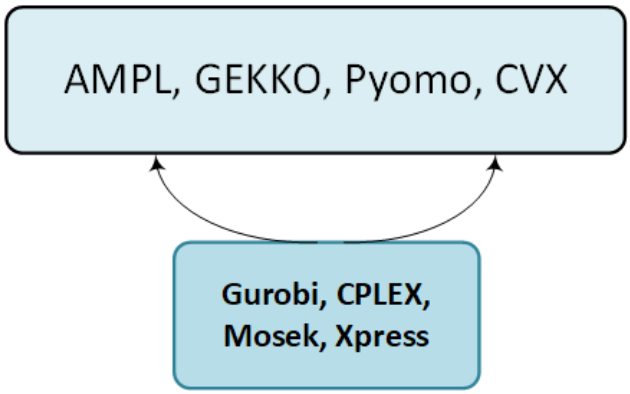

Figure 14: Base optimization engines and high-level engines

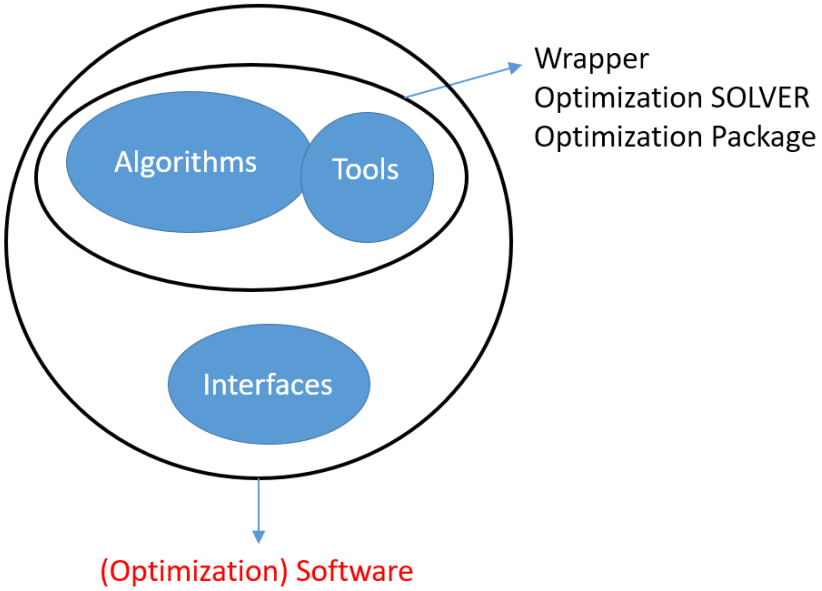

Figure 15: What is a solver?

Why we have chosen Gurobi as an optimization engine? Just Goooogle:

```
gurobi benchmark
```

To install the gurobi, we need to do two things: 1) install gurobi optimizer from `gurobi.com`, 2) install gurobipy package via `pip`:

```
pip install gurobipy
```

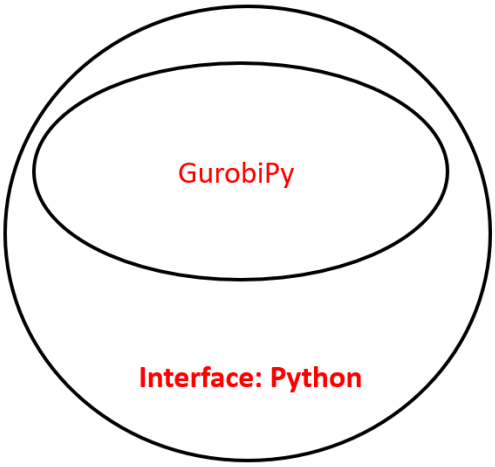

Figure 16: Anatomy of gurobipy

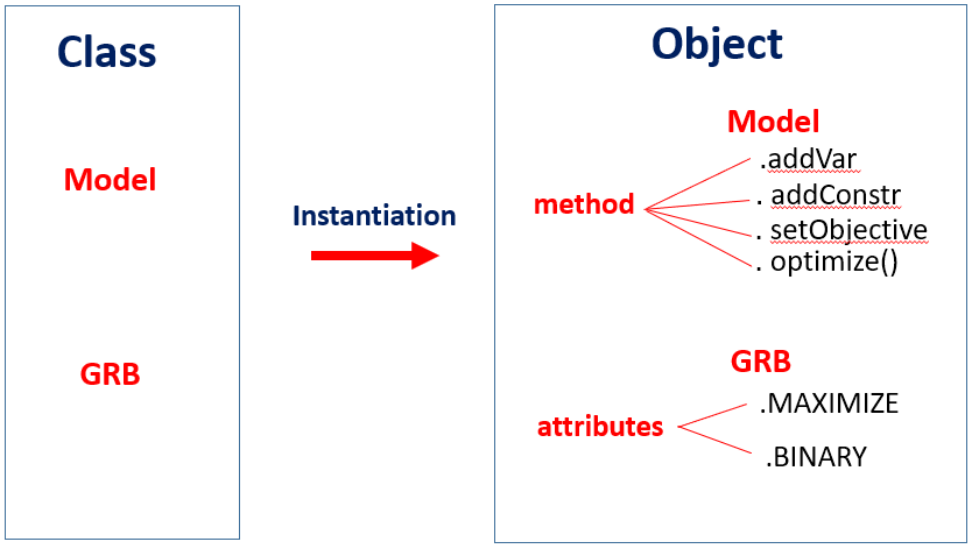

Figure 17: Instantiation of an object through gurobipy classes

Remark. Package Installer for Python (PIP) is the de facto and recommended package-management system written in Python and is used to install and manage software packages. It connects to an online repository of public packages, called the Python Package Index. If you are using JAVA, the package manager is MAVEN.

3.5 Obtaining an Academic Gurobi License

To acquire an academic license, we need to access gurobi website inside an academic institution and run a code on our PC. This license is unique and only works on the computer which that script is executed.

Unfortunately, in Iran, several restrictions have eliminated the ability to get a valid academic license. So, we need to perform two tricks. 1) TAKE CARE OF DNS LEAK, 2) ROUTE OUR IP TO AN ACADEMIC INSTITUTION IN A FREE COUNTRY.

Remark. We provide a short video clip to explain the instruction in detail.

3.6 How to Define Decision Variables, Constraints, and Objective Functions (GurobiPy)

As we said before, to get a complete overview of methods and function of Gurobi API in python, just gooooooooogle:

```
gurobi Python API Overview
```

At the basic level, all you need to know about gurobi python api is the following methods:

```
# to invoke libraries
from gurobipy import Model, GRB

# to define a vairiable (you can google online)
addVar
addVars

# to define a constaint or constraints (be carefull about camel case!)
addCosntr
addConstrs

#to set objective
setObjective

#and finally optimize
optimze()
```

3.7 Modeling Toy Problem (GurobiPy)

Let's develop a simple two-level supply chain model and then code in python:

Example 4 (A location allocation problem or a simple two-level production-distribution supply chain problem)**.** To supply a commodity to customers, it will be first stored at m potential facilities and then be transported to n customers. The fixed cost of the building facilities at site i is f_i and the unit capacity cost is a_i for $i = 1, \ldots, m$. The demand is d_j for $j = 1, \ldots, n$, and the unit transportation cost between i and j is c_{ij}. The maximal allowable capacity of the facility at site i is u_i. Let $y_i \in \{0, 1\}$ be the facility location variable and $x_{ij} \in \mathbb{R}_+$ be the transportation variable. The mixed-integer program of the problem should be sth like this:

$$
\begin{aligned}
\textbf{minimize} \quad & \sum_i f_i y_i + \sum_i \sum_j c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_j x_{ij} \leq u_i y_i, \ \forall i, \\
& \sum_i x_{ij} \geq d_j, \ \forall j, \\
& y_i \in \{0, 1\}, \quad x_{ij} \geq 0, \ \forall i, j.
\end{aligned}
$$

First, we introduce our notation [syntax] in python for variables and inputs:

```
#constants (dimensions)
n_facility
n_customer

#input arrays (parameters)
p_cost_fixed
p_cost_transport
p_capacity_max
p_demand

#variables
v_transport
v_locate
```

Next, we are going to define 1) constants and arrays, 2) variables, 3) constraints, and 4) objective and model sense.

```
import numpy as np
from gurobipy import Model, GRB
```

```
#instantiate class
scm = Model("two-level-SCM")


# just instantiation of paramaters for defining structure and unit-test
# p_cost_fixed = [1,1,1]
# p_cost_transport = np.full((3,4),1)
# p_capacity_max = [1,1,1]
# p_demand = [1,1,1,1]

# you do not need to talk about this real data
# we need to go back and initialize the arrays
p_cost_fixed = np.array([120,150,135])
p_cost_transport = np.array([[10,20,15,16],
[5,17,18,15],
[13,15,18,15]])
p_demand = np.array([15,35,22,41])

# constants
n_facility = p_cost_transport.shape[0]
n_customer = p_cost_transport.shape[1]

# decision variables
v_transport = scm.addVars(n_facility,n_customer)
v_locate = scm.addVars(n_facility, vtype=GRB.BINARY)

#constraint 1
for i in range(n_facility):
scm.addConstr(sum(v_transport[i,j] for j in range(n_customer))
<= p_capacity_max[i]*v_locate[i])

# We can define the first constraint with quicksum
from gurobipy import quicksum
scm.addConstrs(quicksum(v_transport[i,j] for j in range(n_customer))
<= p_capacity_max[i]*v_locate[i] for i in range(n_facility))


#constraint 2
for j in range(n_customer):
scm.addConstr(sum(v_transport[i,j] for i in range(n_facility)) >=
p_demand[j])
```

```
# objective
scm.setObjective(sum(v_locate[i]*p_cost_fixed[i] for
i in range(n_facility)) +
sum(v_transport[i,j]*p_cost_transport[i,j] for i in range(n_facility)
for j in range(n_customer)),GRB.MINIMIZE)

# model sense
scm.ModelSense = GRB.MINIMIZE

scm.update()

scm.optimize()
```

3.8 Extracting and Streaming Results (GurobiPy)

Now, it is possible to retrieve the values of decision variables and the objective function with `getAttr` and `objVal`:

```
#get results from variables
status = scm.status
if status == GRB.OPTIMAL:
print("the objective is: ",scm.objVal)
print(scm.getAttr('X',v_transport))
print(scm.getAttr('X',v_locate))
```

Remark. We can auto-complete the syntax with `TAB` in jupyterlab.

We can create a dictionary of decision variables values:

```
dict_v_transport = {s: v_transport[s].X for s in
         np.ndindex(n_facility,n_customer)}

# get the keys or values
dict_v_transport.keys()
dict_v_transport.values()
```

Or convert it into Pandas Series for a better visualization:

```
import pandas as pd
df = pd.Series(dict_v_transport)
df
```

Final result:

```
0  0      0.0
1      0.0
2      0.0
3      0.0
1  0     15.0
1     35.0
2     22.0
3     41.0
2  0      0.0
1      0.0
2      0.0
3      0.0
dtype: float64
```

Matrix Programming (compact model) in gurobi.

Suppose that we want to code the following OP program:

$$\begin{aligned} \min_{x} \quad & x^T Q x \\ \text{s.t.} \quad & Ax \geq b \\ & x \geq 0. \end{aligned}$$

With the classical syntax we know:

```
# A 3 var problem
from gurobipy import Model,GRB
m = Model()
x1 = m.addVar(ub=1.0)
x2 = m.addVar(ub=1.0)
x3 = m.addVar(ub=1.0)
m.setObjective(x1*x1 + 2*x2*x2 + 3*x3*x3)
m.addConstr(x1 + 2*x2 + 3*x3 >= 4)
m.addConstr(x1 + x2 >= 1)
m.optimize()
```

With the matrix programming format (New after version 9.0):

```
# A 3 var problem
from gurobipy import Model,GRB
import numpy as np
```

```
m = Model()
Q = np.diag([1, 2, 3])
A = np.array([ [1, 2, 3], [1, 1, 0] ])
b = np.array([4, 1])
x = m.addMVar(3, ub=1.0)
m.setObjective(x @ Q @ x)
m.addConstr(A @ x >= b)
m.optimize()
```

In the next chapter, we are going to introduce more advanced methods like **getConstrByName** for iterative solving an optimization model.

There are numerous gurobi communities that we can discuss and share our thoughts and issues:

```
https://or.stackexchange.com/
https://groups.google.com/g/gurobi
```

4 Robust Optimization

4.1 Motivation to Robust Optimization

Classical Approach on Handling Uncertainty. We handle uncertainty through three major frameworks: Stochastic optimization starts by assuming the uncertainty has a probabilistic description. The motivation for this approach is twofold. First, the model of set-based uncertainty is interesting in its own right, and in many applications is an appropriate notion of parameter uncertainty. Second, computational tractability is also a primary motivation and goal. However, fuzzy programming is based on the expert opinion and thus it is subjective.

Table 1: Frameworks based on the level of data accessibility

Framework	Origin	Data-accessibility
Stochastic Programming	Pobability Theory, Stochastic Processes	Normal
Flexible Programming	Fuzzy Programming	Expert-based Data (may know the occurrence region higher than other region)
Robust Optimization	**Control Theory**, Stochastic Process	Only Support

Table 2: Frameworks based on the tools

Framework	Tools
Stochastic Programming	Chance Constrained, Value at Risk (VaR), Sample Average Approximation (SAA)
Flexible Programming	Possibility, Necessity, Credibility
Robust Optimization	Uncertainty Sets

Figure 18: Levels of data accessibility

- Case 1. We have collected a data-set (historical data) and its occurrence and we have fit a probability distribution with a high confidence. We know the depth and the support data with a high confidence; e.g., a news-vendor problem in which the demand data comes from the last month sale.
- Case 2. We know the support but the depth is not through (imperfect historical data) and it is derived form expert's opinion; e.g., the temperature of a furnace assuming that we don not have any probe.

- Case 3. We only know the support; e.g., a truck driver that only knows he drives not less than 60 km/h and not more than 110 km/h. As you can see, collecting data about depth is not easy or unnecessary.

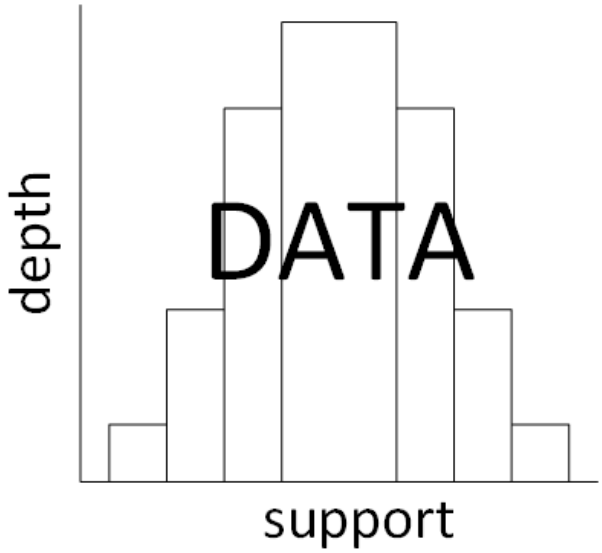

Figure 19: Data: support vs. depth

4.2 Understanding Information Base, Realization Mechanisms, and Uncertainty Sets

So, at this moment, we only focus on set-based uncertintay or .....

The key components of an uncertainty set are: nominal values of uncertain parameters, perturbation values (thus creates us the information base), and the realization mechanism. Consider the following stochastic model,

$$\begin{aligned} \max_{x} \quad & f(x) \\ \text{s.t.} \quad & \sum_{j=1}^{n} \tilde{a}_{ij} x_j \leq b_i, \quad \forall i = 1, \ldots, m, \\ & x \in \mathbb{R}_+, \end{aligned} \tag{3}$$

specifically, the information base is as follows:

$$\tilde{a}_{ij} : [a_{ij} - \hat{a}_{ij} = a^l, a_{ij} + \hat{a}_{ij} = a^u] \xmapsto{\textbf{p.m.}} \xi_{ij} : [-1, +1] \tag{4}$$

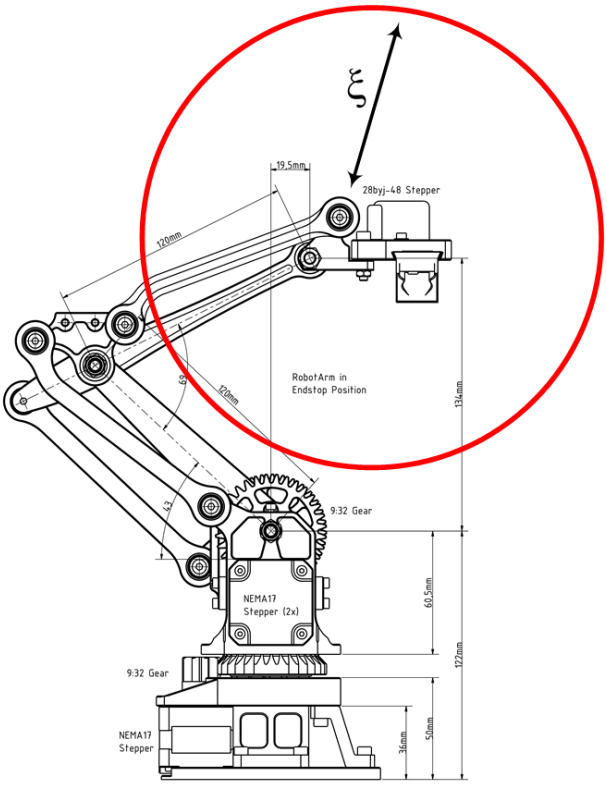

Figure 20: Motivation to Robust Optimization

or equally,

$$\bar{a} = \frac{a^l + a^u}{2},\ \hat{a} = \frac{a^u - a^l}{2}. \tag{5}$$

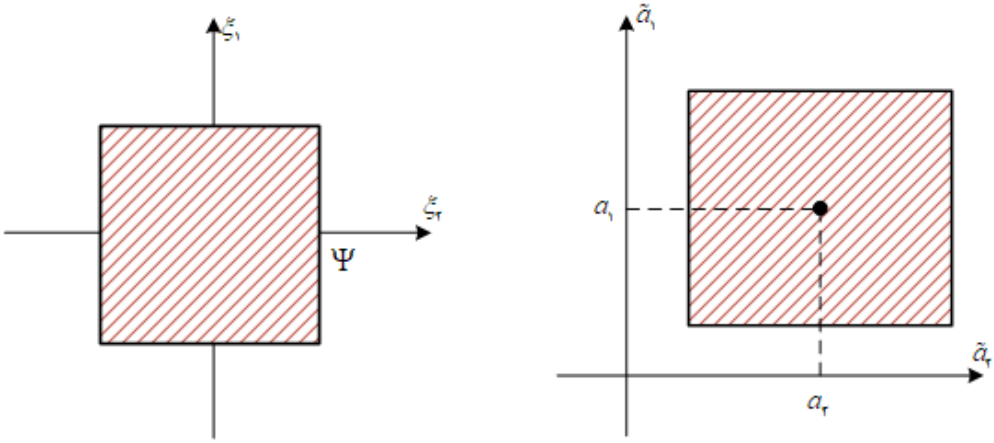

Figure 21: Illustration of information base (two stochastic parameters)

where $f(x)$ is a concave function in x. Parameters $\tilde{a}_{ij}$ and $\tilde{b}_i$ are not deterministic and

their respective stochastic states are oscillating within a priori threshold, i.e., *support*, with unknown probability functions. Assume that $\tilde{a}_{ij} : [a_{ij} - \hat{a}_{ij}, a_{ij} + \hat{a}_{ij}] \xrightarrow{\textbf{p.m.}} \xi_{ij} : [-1, +1]$ where **p.m.** stands for *parametrized mapping*. We also assume that the discrete support of the stochastic variable produces a closed, finite, and countable set. Then, we apply the worst-case on uncertainty-affecting parts:

$$\begin{aligned} \max_{x} \quad & f(x) \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_{ij}x_j + \max_{\xi_{ij} \in \mathcal{U}} \left\{ \sum_{j=1}^{n} \hat{a}_{ij}\xi_{ij}x_j \right\} \leq b_i, \quad \forall i = 1, \ldots, m, \qquad (6) \\ & x \in \mathbb{R}_+. \qquad (7) \end{aligned}$$

Remark. What is exactly a norm function? For a vector x, the associated norm is written as $\|x\|_p$ or l_p where p is some value. The value of the norm of x with some length n is as follows,

$$\|x\|_p = \left(x_1^p + x_2^p + \ldots + x_n^p\right)^{\frac{1}{p}}$$

1. One-norm is the sum of absolute values

$$\|x\|_1 = |x_1| + |x_2| + \ldots + |x_n|$$

2. Euclidean norm (also called L2-norm)

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$$

3. Maximum norm is the maximum absolute value

$$\|x\|_\infty = \max\{x_1, x_2, \ldots, x_n\}$$

Next, we need to pick our uncertainty set:

1. box (inf-norm) uncertainty set

$$\mathcal{U}_\infty = \left\{ \xi \,|\, \|\xi\|_\infty \leq \Psi \right\}$$

2. ellipsoidal (2-norm) uncertainty set

$$\mathcal{U}_2 = \left\{ \xi \,|\, \|\xi\|_2 \leq \Omega \right\}$$

3. disciplined polyhedral (1-norm) uncertainty set

$$\mathcal{U}_1 = \left\{ \xi \,|\, \|\xi\|_1 \leq \Gamma \right\}$$

4. budgeted uncertainty set

$$\mathcal{U}_i^{\Gamma} = \left\{ S_i \cup \{t_i\} \,|S_i \subseteq J_i,\ |S_i| = \lfloor \Gamma_i \rfloor\ ,\ t_i \in J_i \setminus S_i \right\}$$

Suppose that there are 10 stochastic parameters in row i ($|J_i|$) , and budget of uncertainty is 7.5 ($\Gamma_i = 7.5$); hence, 7 stochastic parameters should take their worst-case values ($|S_i| = \lfloor \Gamma_i \rfloor$), and one stochastic parameter should take the half ot its worst-case ($(\Gamma_i - \lfloor \Gamma_i \rfloor)\, \hat{a}_{ij}$).

4.3 General Strategies to Derive Robust Counterparts

There are numerous ways to derive a robust counterpart, i.e., to transform the inner max statement into a explicit statement and then plug in the constraint:

1. Conic Reformulation and Conic Duality
2. Lagrangian Dual
3. Farkas Lemma and Strong Duality
4. Norm Spaces and Dual norm

We are going to derive the dual form of inner maxes via the Dual Norm properties. Prior to introducing these properties, let us transform the inner max statement into the vector form: $\xi_i^T\,(\hat{a}_i x)$ or

$$\begin{bmatrix} \xi_{i1} \\ \xi_{i2} \\ \vdots \\ \xi_{in} \end{bmatrix} \begin{bmatrix} \hat{a}_{i1}x_1 & \hat{a}_{i2}x_2 & \cdots & \hat{a}_{in}x_n \end{bmatrix} = \sum_{j=1}^{n} \hat{a}_{ij}\xi_{ij}x_j$$

Theorem 4.1. *The dual of the p-norm is the q-norm, where q satisfies* $\frac{1}{p} + \frac{1}{q} = 1$, *that is,* $q = \frac{p}{p-1}$.

Theorem 4.2. *The dual of inner max function w.r.t. uncertainty set* $\mathcal{U}_p = \left\{ \xi \,\middle|\, \|\xi\|_p \leq \Delta \right\}$ *is*

$$\textit{Dual} \left\{ \max_{\xi_i \in \mathcal{U}_p} \left\{ \xi_i^T\,(\hat{a}_i x) \right\} \right\} = \Delta \|\hat{a}_i x\|_q$$

Example 5. We are going to derive the robust counterpart regarding l_2 norm.

$$\texttt{Dual} \left\{ \max_{\xi_{ij} \in \mathcal{U}_2} \left\{ \xi_i^T\,(\hat{a}_i x) \right\} \right\} = \Omega \|\hat{a}_i x\|_2 = \Omega \sqrt{\sum_{j=1}^{n} \hat{a}_{ij}^2 x_j^2}$$

since $p = 2$ then $q = 2$. Consequently, for l_∞ the dual of the inner max is $\Psi \sum_{j=1}^{n} \left|\hat{a}_{ij} x_j\right|$ and for l_1 ($q = \frac{1}{1-1} = \infty$) is

$$\begin{aligned} & \text{subject to:} \\ & \sum_{j=1}^{n} a_{ij} x_j + \Gamma \lambda_i \leq b_i, \ \forall i \\ & \lambda_i \geq \left|\hat{a}_{ij} x_j\right|, \forall i, \forall j \\ & \lambda_i \geq 0, \forall i \end{aligned} \tag{8}$$

Example 6. Consider the following LP with uncertain parameters.

$$\begin{aligned} \max_{x} \quad & \tilde{c}_1 x_1 + \tilde{c}_2 x_2 \\ \text{s.t.} \quad & \tilde{a}_1 x_1 + \tilde{a}_2 x_2 \leq \tilde{b} \\ & 2x_1 + 3x_2 \leq 7 \\ & x \in \mathbb{R}_+. \end{aligned}$$

And the support of the parameters are:

$$\begin{aligned} & \tilde{c}_1 \in [0.5, 1.5] \quad \tilde{c}_2 \in [1, 2.5] \\ & \tilde{a}_1 \in [1, 3] \qquad \tilde{a}_2 \in [1, 2] \\ & \tilde{b} \in [5, 7] \end{aligned}$$

Therefore, the robust counterpart w.r.t. l_∞ is:

$$\begin{aligned} \max_{x,z} \quad & z \\ \text{s.t.} \quad & x_1 + 1.75x_2 + \Psi_1 \left(0.5|x_1| + 0.75|x_2|\right) \geq z \\ & 2x_1 + 1.5x_2 + \Psi_2 \left(1 + |x_1| + 0.5|x_2|\right) \leq 6 \\ & 2x_1 + 3x_2 \leq 7 \\ & x \in \mathbb{R}_+, z \in \mathbb{R}. \end{aligned}$$

And we can conveniently drop the absolute signs.

Theorem 4.3 (robust counterpart w.r.t. budgeted uncertainty set)**.** *Again, suppose the following LP with uncetain parameter:*

$$\max_{x} \quad f(x)$$

$$s.t. \quad \sum_{j=1}^{n} a_{ij} x_j + \max_{\xi_{ij} \in \mathcal{U}} \left\{ \sum_{j=1}^{n} \hat{a}_{ij} \xi_{ij} x_j \right\} \leq b_i, \quad \forall i = 1, \ldots, m, \tag{9}$$

$$x \in \mathbb{R}. \tag{10}$$

We isolate the inner-max function:

$$\begin{aligned}
\max_{\xi} \quad & \sum_{j\in J_i} \hat{a}_{ij}\xi_{ij}|x_j^*| \\
s.t. \quad & \sum_{j\in J_i} \xi_{ij} \le \Gamma_i, \quad \forall i = 1,\dots,m, \quad \longmapsto z_i(\textbf{\textit{dual}}) \\
& 0 \le \xi_{ij} \le +1. \quad \longmapsto p_{ij}(\textbf{\textit{dual}})
\end{aligned}$$

and the dual is:

$$\begin{aligned}
\min_{p,z} \quad & \Gamma_i z_i + \sum_{j\in J_i} p_{ij} \\
s.t. \quad & z_i + p_{ij} \ge \hat{a}_{ij}|x_j^*|, \quad \forall i, \forall i \in J_i \\
& p, z \in \mathbb{R}_+.
\end{aligned}$$

and finally we plug the dual in the inner-max and we get the robust counterpart (by strong duality since the primal and the dual are both bounded and feasible):

$$\begin{aligned}
\max_{x,p,y,z} \quad & f(x) \\
s.t. \quad & \sum_{j\in J_i} a_{ij}x_j + z_i\Gamma_i + \sum_{j\in J_i} p_{ij} \le b_i, \quad \forall i, \\
& z_i + p_{ij} \ge \hat{a}_{ij}y_j, \quad \forall i, j \in J_i \\
& -y_j \le x_j \le y_j, \quad \forall j \\
& x, p, y, z \in \mathbb{R}_+.
\end{aligned}$$

4.4 Flashback: GurobiPy Iterating Model and [Reliability, Conservatism, and Over-protection]

In this section, first we derive a robust counterpart of a knapsack problem w.r.t. budgeted uncertainty set and then we discuss the budget sensitivity by coding the robust counterpart in python [gurobipy].

Example 7. The knapsack problem is a problem in combinatorial optimization: Given a set of items, each with a weight (w) and a utility (u), determine which item should be selected (x) to include in a collection so that the total weight ($\sum_{j=1}^{n} w_j x_j$) is less than or equal to a given limit (C) and the total utility ($\sum_{j=1}^{n} u_j x_j$) is as large as possible. Weights $\tilde{w}_j$ are not deterministic and their respective stochastic states are oscillating within $\tilde{w}_j : [w_j - \hat{w}_j, w_j + \hat{w}_j]$. Note that we have only one constraint!

$$\begin{aligned} \max_{x} \quad & \sum_{j=1}^{n} u_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^{n} \tilde{w}_j x_j \leq C, \\ & x \in \{0,1\}. \end{aligned}$$

Now, we apply the budgeted uncertainty set to the worst-case part:

$$\begin{aligned} \max_{x} \quad & \sum_{j=1}^{n} u_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^{n} w_j x_j + z\Gamma + \sum_{j=1}^{n} p_j \leq C, \\ & z + p_j \geq \hat{w}_j y_j, \quad \forall j \in \{1,2,\ldots,n\} \\ & -y_j \leq x_j \leq y_j, \quad \forall j \in \{1,2,\ldots,n\} \\ & x \in \{0,1\}, \\ & y, z, p \in \mathbb{R}_+. \end{aligned} \tag{11}$$

First, let us introduce our notation [syntax] in python for variables and inputs:

```
#constants (dimensions)
n_items

#input arrays (parameters)
p_utility
p_weight
p_Gamma
p_Capacity

#set
s_item

#variables
v_select
v_dual_z
v_dual_p
v_dual_y
```

List of works we are going to do:

1. One-time Optimization: We are going to define 1) constants and arrays, 2) variables, 3) constraints, and 4) objective and model sense.
2. Iterative optimization over different values of Gamma by `getConstrByName` method. Be careful about how we are going to add constraint, then remove it, and then update the instance of the model over different iterations.

```
import numpy as np
from gurobipy import Model, GRB

#input arrays (parameters)
p_utility = [26,65,35,48,26,53,32,48,58,62]
p_weight = [27,28,16,28,24,25,19,23,29,29]
p_Gamma = 0 # deterministic
p_Capacity = 120

n_items = len(p_utility)

s_item = range(n_items)

model = Model("robust_knapsack")

v_select = model.addVars(n_items, vtype=GRB.BINARY)
v_dual_p = model.addVars(n_items)
v_dual_y = model.addVars(n_items)
v_dual_z = model.addVar()

p_weight_hat = [p_weight[j]*.1 for j in s_item]

model.setObjective(sum(v_select[j] * p_utility[j] for j in s_item), GRB.MAXI

model.addConstr(sum(v_select[j] * p_weight[j] for j in s_item) +
p_Gamma*v_dual_z + sum(v_dual_p[j] for j in s_item) <= p_Capacity)

for j in s_item:
model.addConstr(v_select[j] >= -1 * v_dual_y[j]) #straightforward
model.addConstr(v_select[j] <=  v_dual_y[j])

for j in s_item:
model.addConstr(v_dual_p[j] + v_dual_z >= p_weight_hat[j] * v_select[j])
```

```
model.optimize()


status = model.status
        if status == GRB.OPTIMAL:
        print("the objective is: ",model.objVal)
        print(model.getAttr('X',v_select))

v_select_dict = [v_select[s].X for s in range(n_items)]

print(np.array(v_select_dict).astype(int))


# so I want to change the Gamma value iteratively and capture the objective
obj_vec = np.zeros(n_items)

model.setParam('OutputFlag', 0)

for p_Gamma in s_item:
        model.addConstr(sum(v_select[j] * p_weight[j] for j in s_item)
        + p_Gamma*v_dual_z + sum(v_dual_p[j] for j in s_item)
        <= p_Capacity,'cap_constraint')
        model.optimize()
        print("model with Gamma = ",p_Gamma," is optimized !")
        obj_vec[p_Gamma] = model.objVal
        model.remove(model.getConstrByName('cap_constraint'))
        model.update()

print(obj_vec)

# now we want to undersiand the optimal protection level, conservatism,
# and over-protection.
import matplotlib.pyplot as plt
fig = plt.figure()
plt.plot(obj_vec)
plt.ylabel('objective')
plt.xlabel('GAMMA')
plt.grid()
fig.set_size_inches(10,6)
plt.show()
```

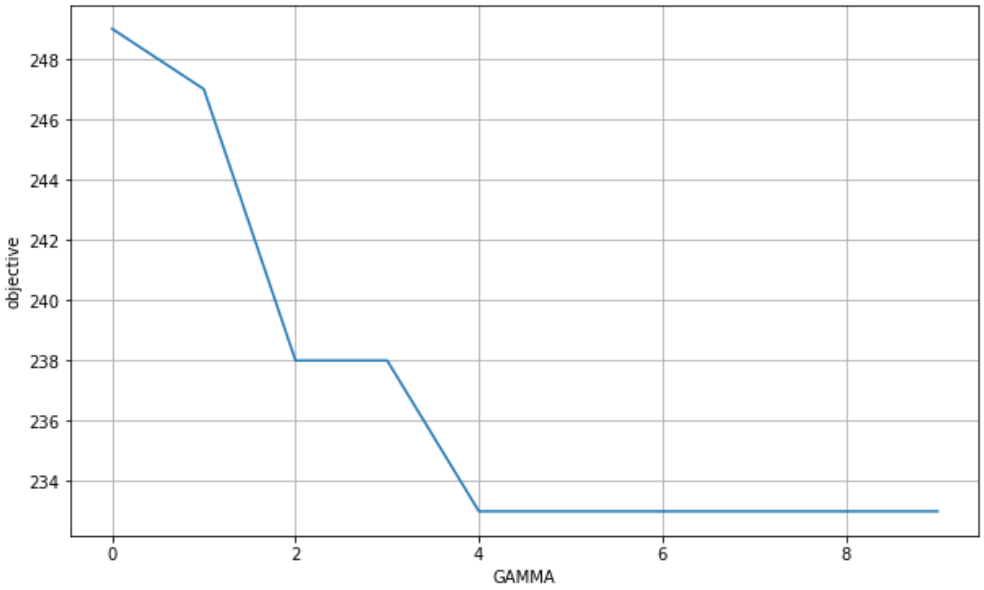

Figure 22: Solutions to the knapsack problem w.r.t. different budgets

Next, we are going to calculate the **constraint violation**:. This time with gamma step = 0.5.

```
import numpy as np
from gurobipy import Model, GRB
#input arrays (parameters)
p_utility = [26,65,35,48,26,53,32,48,58,62]
p_weight = [27,28,16,28,24,25,19,23,29,29]
p_Gamma = 0 # deterministic
p_Capacity = 120
#constants (dimensions)
n_items = len(p_utility)
s_item = range(n_items)
model = Model("robust_knapsack")
v_select = model.addVars(n_items, vtype=GRB.BINARY)
v_dual_p = model.addVars(n_items)
v_dual_y = model.addVars(n_items)
v_dual_z = model.addVar()
p_weight_hat = [p_weight[j]*.1 for j in s_item]
model.setObjective(sum(v_select[j] * p_utility[j] for j in s_item), GRB.MAXI
for j in s_item:
model.addConstr(v_select[j] >= -1 * v_dual_y[j])
model.addConstr(v_select[j] <=  v_dual_y[j])
for j in s_item:
```

```
model.addConstr(v_dual_p[j] + v_dual_z >= p_weight_hat[j] * v_select[j])

#  Gamma steps =.5

obj_vec = np.zeros(n_items*2) # this why i am multipying with 2 (should be c
p_Gamma_vec = np.arange(0,n_items,.5).tolist() # should be corrected
model.setParam('OutputFlag', 0)
violation_perc = 0
for g in range(n_items*2): # should be corrected
        model.addConstr(sum(v_select[j] * p_weight[j] for j in s_item) +
         p_Gamma_vec[g]*v_dual_z + sum(v_dual_p[j] for j in s_item)
         <= p_Capacity,'cap_constraint')
        model.optimize()
        v_select_dict = [v_select[s].X for s in range(n_items)]
        v_select_array_int = np.array(v_select_dict).astype(int)
        lhs = np.sum(v_select_array_int * np.array(p_weight))
        if lhs - p_Capacity < 0:
                violated = ''
                violation_perc = (p_Capacity - lhs)*100/p_Capacity
        else:
                violated = 'not'
        print("model with Gamma = ",p_Gamma_vec[g]," is optimized !
         and the Constraint is",violated,"violated.
          (violation percent = ",violation_perc,")")
        obj_vec[g] = model.objVal
        model.remove(model.getConstrByName('cap_constraint'))
model.update()
del model
```

and the output will be:

```
model with Gamma =  0.0  is optimized ! and the Constraint is not violated.
 (violation percent =  0 )
model with Gamma =  0.5  is optimized ! and the Constraint is  violated.
 (violation percent =  2.5 )
model with Gamma =  1.0  is optimized ! and the Constraint is  violated.
 (violation percent =  2.5 )
model with Gamma =  1.5  is optimized ! and the Constraint is  violated.
 (violation percent =  4.166666666666667 )
model with Gamma =  2.0  is optimized ! and the Constraint is  violated.
 (violation percent =  7.5 )
model with Gamma =  2.5  is optimized ! and the Constraint is  violated.
```

```
 (violation percent =  7.5 )
model with Gamma =  3.0  is optimized ! and the Constraint is  violated.
 (violation percent =  7.5 )
model with Gamma =  3.5  is optimized ! and the Constraint is  violated.
 (violation percent =  9.166666666666666 )
model with Gamma =  4.0  is optimized ! and the Constraint is  violated.
 (violation percent =  9.166666666666666 )
model with Gamma =  4.5  is optimized ! and the Constraint is  violated.
 (violation percent =  9.166666666666666 )
model with Gamma =  5.0  is optimized ! and the Constraint is  violated.
 (violation percent =  9.166666666666666 )
model with Gamma =  5.5  is optimized ! and the Constraint is  violated.
 (violation percent =  9.166666666666666 )
model with Gamma =  6.0  is optimized ! and the Constraint is  violated.
 (violation percent =  9.166666666666666 )
model with Gamma =  6.5  is optimized ! and the Constraint is  violated.
 (violation percent =  9.166666666666666 )
model with Gamma =  7.0  is optimized ! and the Constraint is  violated.
 (violation percent =  9.166666666666666 )
model with Gamma =  7.5  is optimized ! and the Constraint is  violated.
 (violation percent =  9.166666666666666 )
model with Gamma =  8.0  is optimized ! and the Constraint is  violated.
 (violation percent =  9.166666666666666 )
model with Gamma =  8.5  is optimized ! and the Constraint is  violated.
 (violation percent =  9.166666666666666 )
model with Gamma =  9.0  is optimized ! and the Constraint is  violated.
 (violation percent =  9.166666666666666 )
model with Gamma =  9.5  is optimized ! and the Constraint is  violated.
 (violation percent =  9.166666666666666 )
```

So, one remaining question? How can we spot the parameter which took its perturbation part? Consider $\Gamma = 1.5$. We choose $\Gamma = 1.5$ since its violation percentage is the closest to 5 percent violation (and thus 95 percent **protection level**). We again solve the problem with $\Gamma = 1.5$ and obtain the optimal value of x. Then we compute the following vector from 11:

$$\hat{w}x^*$$

By sorting the above vector, the last parameter in the sorted vector gets the full budget and the second parameter form the end gets the half of the budget.

```
x = [0,1,1,0,0,0,1,1,0,1]
perturb_vec = np.array(x)*np.array(p_weight_hat)
```

```
print(perturb_vec)
np.argsort(perturb_vec)
```

Output:

```
array([0, 3, 4, 5, 8, 2, 6, 7, 1, 9], dtype=int64)
```

The tenth parameter should be $w_{10} + \hat{w}_{10}$ and the second should be $w_2 + 0.5\hat{w}_2$.

Is it possible to compute all scenarios w.r.t. Γ and then find the best possible violation degree w.r.t. the protection level? No, often we need to guess or estimate the upper bound of the violation degree. Two of the famous upper bounds are as follows (bertsimas):

$$\Pr\left(\sum_j \tilde{a}_{ij}x_j^* > b_i\right) \leq \exp\left(\frac{-\Gamma_i^2}{2|J_i|}\right)$$

and even more tighter:

$$\Pr\left(\sum_j \tilde{a}_{ij}x_j^* > b_i\right) \leq 1 - \phi\left(\frac{\Gamma_i - 1}{\sqrt{n}}\right)$$

Remark. Note that these bounds are completely independent of the values of decision variables.

Therefore, we can plot the Γ vs. upper bound of Violation Percentage. Or, we can simply use `interp` from `numpy` package to find the appropriate value for Γ.

```
import numpy as np
n_items = 10
Gamma = np.arange(0,n_items+.1,.1)
bound1 = np.exp(-Gamma**2/(2*n_items))

from scipy.stats import norm
bound2 = 1 - norm.cdf(Gamma)

import matplotlib.pyplot as plt
fig = plt.figure()

plt.plot(Gamma,bound1,'-',label="bound 1",linewidth=4)
plt.plot(Gamma,bound2,'--',label="bound 2",linewidth=4)
```

```
plt.ylabel('Violation Percent')
plt.xlabel('GAMMA')
plt.grid()
plt.legend()
fig.set_size_inches(10,6)
plt.show()

#or using interpolation
np.interp(.05, bound2[::-1], Gamma[::-1]) # the first arg should be increasi
# output: 1.6468
```

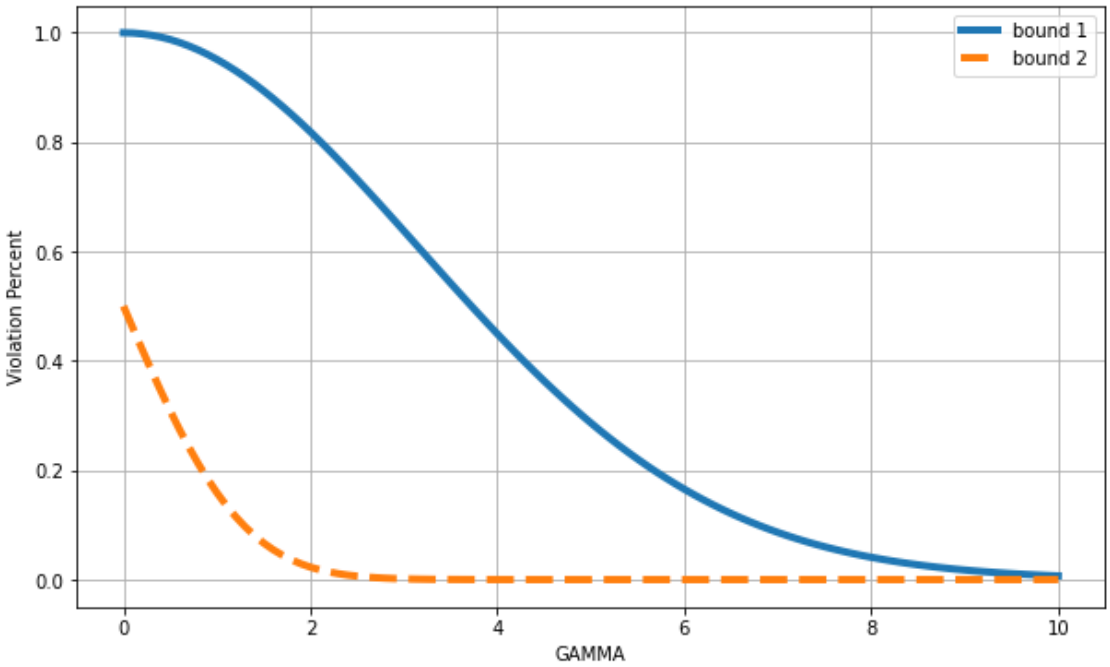

Figure 23: Violation percentages vs. Gamma w.r.t. two bounds

According to results, the second bound is the closest to our analysis (1.6468) w.r.t. violation degree 5 percent.

Remark. `scipy` package contains statistical tools, which is also on top of `numpy` package and preserve the properties of this package, too.

So when you are developing uncertainty set you need to take care of two things:

- The hyper-parameters (like Γ) that control the conservatism, the degree of violation, and the protection level.

- Tractability of robust counterparts.
- The Realization Dynamics (in the next section).

4.5 Data-driven Robust Optimization: Motivation and Taxonomy

There have been plenty of works in the area of data-driven robust optimization often completely independent of each other; each has attempted to introduce his/her version. The keywords for some of major and outstanding works are:

- Data-driven Chance Constrained
- Distributionally Robust Optimization
- Data-driven Stochastic Programming and Wasserstein Metric (a distance function defined between probability distributions)
- Likelihood Uncertainty (Ambiguity) Sets
- Intersection of Robust Optimization and Machine Learning

However, Our Modern View: The uncertainty sets are the heart and the driver of robust models. In data-driven uncertainty programming, we know the support of data as a priori, and by introducing the depth to data [a posteriori], we reduce the support. Thus, we can create more realistic and less-conservative decisions. Thus, we are building a "machine" or a "black box" without exploring the distributional and probabilistic properties of data, we reduce the uncertainty space by a machine learning technique (see Figure 24). Some might ask why we do not seek a model from stochastic programming if there are enough data. The answer is: The underlying distribution of uncertain parameter may be intrinsically complicated and the moment information is not at hand.

We are going to propose and formulate one of the significant works by Feng-U ..., which used an support vector machine (SVM) to cover the realized data in a convex polyhedron.

4.6 Introduction to Machine Learning and Support Vector Machine

For more detailed information, please refer to the following papers:

1. Shang, Chao, Xiaolin Huang, and Fengqi You. "Data-driven robust optimization based on kernel learning." Computers & Chemical Engineering 106 (2017): 464-479.
2. Ben-Hur, Asa, David Horn, Hava T. Siegelmann, and Vladimir Vapnik. "Support vector clustering." Journal of machine learning research 2, no. Dec (2001): 125-137.

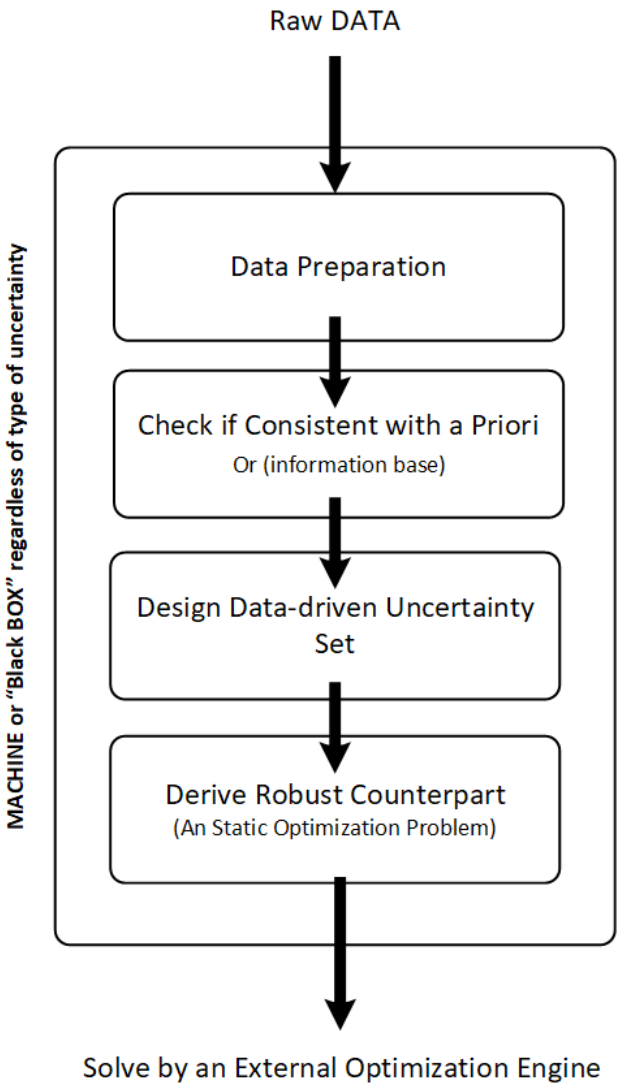

Figure 24: Proposed DDRO framework w.r.t. learning uncertainty set

First, we need to define one thing: What is exactly machine learning. Let me introduce two definitions based on two different views:

- **IBM**. Machine Learning is a branch of **artificial intelligence** and **computer science** which focuses on the use of **data** and **algorithms** to **imitate** the way that human's brains **learn** and **gradually** improving accuracy.
- **A Statistician**. Machine Learning is a **statistical learning process** which consists of statistical non-parametric models, computational geometry, algebra, and topology. And most importantly **Convex Optimization**.

Figure 25 illustrates the properties of a machine (black-box or oracle).

Remark. The core of most machine learning algorithms are an optimization problems.

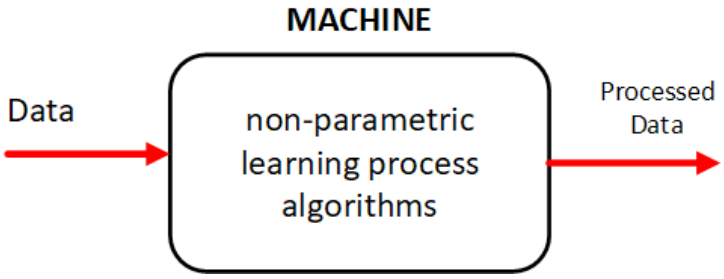

Figure 25: A machine's properties

Support Vector Clustering (SVC) aims to give a description of a number of data points by means of an enclosing sphere with minimal volume (Ben-Hur et al., 2001), thereby enabling an explicit characterization of new data samples that resemble training data.

Consider the following LP with uncertain parameter $\tilde{a}$:

$$\begin{aligned} \max_{x \in X} \quad & f(x) \\ \text{s.t.} \quad & \tilde{a}^T x \leq b. \end{aligned}$$

At time t, we have received a sample vector with size m for $\tilde{a}$:

$$\tilde{a} = \begin{bmatrix} a_1 & a_2 & \cdots & a_m \end{bmatrix}$$

From the viewpoint of machine learning, we now have m **features** and the following is our **feature space** (the dimension of this feature space is m):

$$\tilde{a} = \begin{bmatrix} a_1^{(1)} & a_1^{(2)} & \cdots & a_1^{(n)} \\ a_2^{(1)} & a_2^{(2)} & \cdots & a_2^{(n)} \\ \vdots & \vdots & \ddots & \vdots \\ a_m^{(1)} & a_m^{(2)} & \cdots & a_m^{(n)} \end{bmatrix}$$

Remark. This is support vector **clustering** and we have only one **class**!

Another important term: Mapping Function. A mapping function maps a vector of size m to another vector with size k through a function. Here specially, we are interested in mapping functions that maps current feature space into higher dimension feature spaces. We define the mapping function on input as $\phi(a) : \mathbb{R}^m \to \mathbb{R}^k$. Why? Figure 26 illustrates

the reason behind mapping to a higher dimension. The original 2D feature space is not separable in classes or features. Hence, we map it via the below function to a 3D feature space and it is now linearly separable.

$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}^{m=2} \xrightarrow{\phi\left(\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}\right)} \begin{bmatrix} a_1^2 \\ a_1 a_2 \sqrt{2} \\ a_2^2 \end{bmatrix}^{k=3} \tag{12}$$

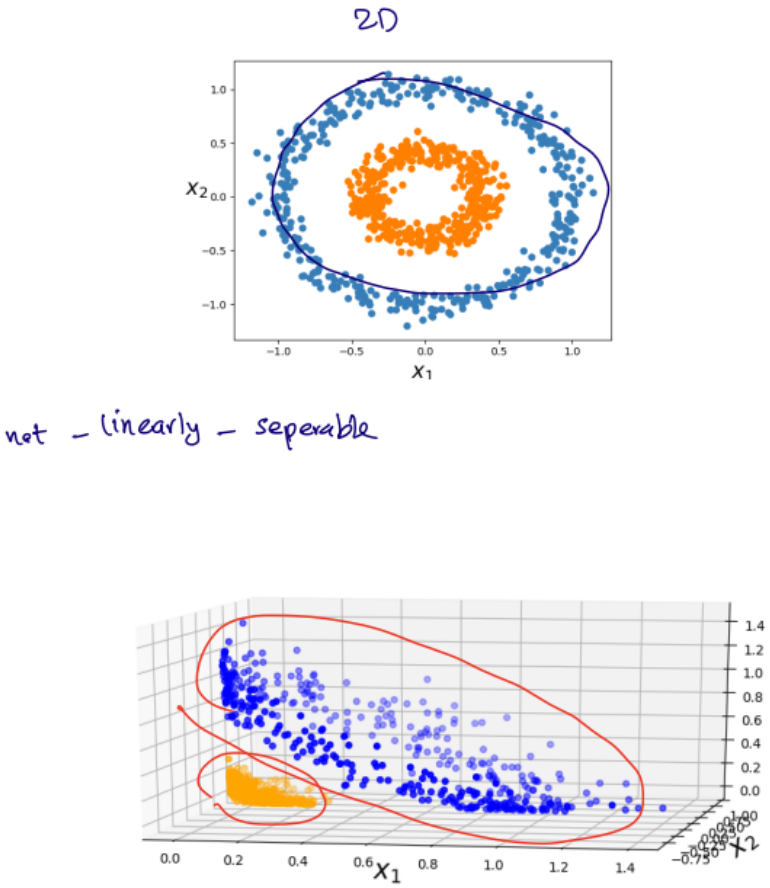

Figure 26: An example of mapping a feature space to higher dimension

Back to SVC, we are going to enclose data in the transformed feature space in a sphere with

minimal volume (**hard-margin SVC**):

$$\begin{aligned} \min_{R,p} \quad & R^2 \\ \text{s.t.} \quad & \left\| \phi\left(a^{(i)}\right) - p \right\|_2^2 \leq R^2, \quad \forall i \in \{1, \cdots, n\}. \end{aligned}$$

Often, we want to control the outliers and how many samples should be placed into the sphere (**soft-margin SVC**):

$$\begin{aligned} P_1: \min_{R,p,\epsilon} \quad & R^2 + \frac{1}{n\nu} \sum_{i=1}^{n} \epsilon_i \\ \text{s.t.} \quad & \left\| \phi\left(a^{(i)}\right) - p \right\|^2 \leq R^2 + \epsilon_i, \quad \forall i \in \{1, \cdots, n\} \longmapsto \alpha_i \text{ (**dual**)}, \\ & \epsilon_i \geq 0, \quad \forall i \in \{1, \cdots, n\} \longmapsto \beta_i \text{ (**dual**)}. \end{aligned}$$

where $\nu \in (0, 1)$ is the regularization parameter and variable ϵ controls the boundary region.

Remark. Here, we assumed that we have previously selected the features. However, an important process in **deep learning** or simply **machine learning** is **feature selection** or **feature engineering**, in which several works are done to select the best possible features (describing the problem accurately) from an **input space**. One of the practical approaches to reduce the feature space is **Principal Component Analysis** or simply PCA.

In order to solve problem (P_1) efficiently, we need to simplify it into a disciplined convex optimization model. First, we formulate the dual of (P_1) as follows (via developing a Lagrangian function):

$$\mathcal{L}(p, R, \epsilon, \alpha, \beta) = R^2 + \frac{1}{n\nu} \sum_{i=1}^{n} \epsilon_i - \sum_{i=1}^{n} \alpha_i \left(R^2 + \epsilon_i - \left\| \phi\left(a^{(i)}\right) - p \right\|^2 \right) - \sum_{i=1}^{n} \beta_i \epsilon_i$$

Next, I'm going to apply the Karush-Kuhn-Tucker condition (KKT) to the Lagrangian function:

$$\frac{\partial \mathcal{L}}{\partial R} = 0 \quad \Rightarrow \quad \sum_{i=1}^{n} \alpha_i = 1 \tag{L1}$$

$$\frac{\partial \mathcal{L}}{\partial p} = 0 \quad \Rightarrow \quad \sum_{i=1}^{n} \alpha_i \left(\phi\left(a^{(i)}\right) - p \right) = 0 \quad \Rightarrow \quad p = \sum_{i=1}^{n} \alpha_i \phi\left(a^{(i)}\right) \tag{L2}$$

$$\frac{\partial \mathcal{L}}{\partial \epsilon_i} = 0 \quad \Rightarrow \quad \sum_{i=1}^{n} \frac{1}{n\nu} - \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \beta_i = 0 \quad \Rightarrow \quad \alpha_i + \beta_i = \frac{1}{n\nu} \tag{L3}$$

Remark. Note that if $f(y) = \frac{1}{2}\|y\|_2^2$, then $\frac{\partial f}{\partial y} = y$.

Next, we insert the results form equations L1, L2, and L3 in the Lagrangian function.

$$\mathcal{L} = \overbrace{R^2}^{A1} + \overbrace{\frac{1}{n\nu}\sum_{i=1}^{n}\epsilon_i}^{A2} - \overbrace{R^2\sum_{i=1}^{n}\alpha_i}^{A3} - \overbrace{\sum_{i=1}^{n}\alpha_i\epsilon_i}^{A4} + \overbrace{\sum_{i=1}^{n}\alpha_i\left\|\phi\left(a^{(i)}\right)-p\right\|^2}^{A5} - \overbrace{\sum_{i=1}^{n}\beta_i\epsilon_i}^{A6}$$

$$\begin{aligned}
A4\ \&\ A6 \quad &\Rightarrow \quad -\sum_{i=1}^{n}\epsilon_i\left(\alpha_i+\beta_i\right) \overset{L3}{=} -\frac{1}{n\nu}\sum_{i=1}^{n}\epsilon_i \\
A2\ \&\ A7 \quad &\Rightarrow \quad -\frac{1}{n\nu}\sum_{i=1}^{n}\epsilon_i + \frac{1}{n\nu}\sum_{i=1}^{n}\epsilon_i = 0 \\
A1\ \&\ A3 \quad &\Rightarrow \quad R^2 - R^2\sum_{i=1}^{n}\alpha_i = R^2\left(\overbrace{1-\sum_{i=1}^{n}\alpha_i}^{0}\right) \overset{L1}{=} 0
\end{aligned} \tag{A7}$$

and finally,

$$\mathcal{L} = A5 \overset{L2}{=} \sum_{i=1}^{n}\alpha_i\left\|\phi\left(a^{(i)}\right) - \overbrace{\sum_{i=1}^{n}\alpha_j\phi\left(a^{(j)}\right)}^{p}\right\|^2$$

For simplification purpose, we denote $\phi\left(a^{(i)}\right)$ and $\phi\left(a^{(j)}\right)$ as ϕ_i and ϕ_j, respectively. Finally, we get the following disciplined QP subject to L1 and L3 (Wolfe Dual):

$$\begin{aligned}
P_2: \ \min_{\alpha} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j\phi_i^T\phi_j - \sum_{i=1}^{n}\alpha_i\phi_i^T\phi_i \\
\text{s.t.} \quad & \sum_{i=1}^{n}\alpha_i = 1, \\
& 0 \le \alpha_i \le \frac{1}{n\nu}, \quad \forall i \in \{1,\cdots,n\}.
\end{aligned}$$

Note that computing $\phi_i^T\phi_j$ is a daunting task, since we need to map the feature space into higher dimension and then calculate the inner product of ϕ_i and ϕ_j. So, here we apply a trick called **kernel** trick (or mercer kernel or kernelization).

Remark. What is Kernelization? In computer science, a kernelization is a technique for designing efficient algorithms that achieve their efficiency by a preprocessing stage in which

inputs to the algorithm are replaced by a smaller input, called a "kernel". The result of solving the problem on the kernel should either be the same as on the original input, or it should be easy to transform the output on the kernel to the desired output for the original problem.

Example 8. Let's apply this concept with or without using this trick. Assume that $\phi : \mathbb{R}^2 \to \mathbb{R}^4$ and

$$\begin{aligned} a &= [a_1, a_2] \quad \phi(a) = \begin{bmatrix} a_1^2 & a_1 a_2 & a_2 a_1 & a_2^2 \end{bmatrix} \\ b &= [b_1, b_2] \quad \phi(b) = \begin{bmatrix} b_1^2 & b_1 b_2 & b_2 b_1 & b_2^2 \end{bmatrix} \end{aligned}$$

We want to calculate $\phi(a) \cdot \phi(b)^T$. Consider the below numerical example:

$$\begin{aligned} &a = [1, 2] \quad \phi(a) = \begin{bmatrix} 1 & 2 & 2 & 4 \end{bmatrix} \\ &b = [3, 4] \quad \phi(b) = \begin{bmatrix} 9 & 12 & 12 & 16 \end{bmatrix} \\ &\textbf{then} \\ &\qquad \phi(a)^T \cdot \phi(b) \overset{\textbf{component-wise}}{=} 9 + 24 + 24 + 64 = 121 \end{aligned} \tag{13}$$

Instead, we apply the 2nd degree kernel on a and b:

$$k\left(a, b\right) = \left([1, 2], [3, 4]^T\right) = (1 \cdot 3 + 2 \cdot 4)^2 = (11)^2 = 121$$

We list some of famous kernels:

- polynomial kernel (d means degree)

$$k(a, b) = (a \cdot b + 1)^d$$

- gaussian kernel

$$k(a, b) = \exp\left(\frac{-\|a - b\|^2}{2\sigma^2}\right)$$

- gaussian radial basis function (RBF)

$$k(a, b) = \exp\left(-\gamma \|a - b\|^2\right)$$

$$\gamma > 0$$

- sigmoid kernel

$$k(a, b) = \tanh\left(ua \cdot b + v\right)$$

Finally, we revise P_1 w.r.t. the kernelized representation, i.e., $\mathbf{k}\left(a^{(i)}, a^{(j)}\right) = \phi_i^T \phi_j$:

$$
\begin{aligned}
P_3: \min_{\alpha} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} \mathbf{k}\left(a^{(i)}, a^{(j)}\right)\alpha_i\alpha_j - \sum_{i=1}^{n}\mathbf{k}\left(a^{(i)}, a^{(i)}\right)\alpha_i \\
\text{s.t.} \quad & \sum_{i=1}^{n}\alpha_i = 1, \\
& 0 \le \alpha_i \le \frac{1}{n\nu}, \quad \forall i \in \{1, \cdots, n\}.
\end{aligned}
$$

Now, we need to distinguish between the points residing on the boundary of the sphere (boundary support vector or BSV) and the points that are inside the sphere (support vector or SV).

If point i is an outlier, then the first constraint of P(1) is completely binding and therefore $\alpha_i > 0$. If point i is on the boundary of the sphere, then first constraint of P(1) is binding ($\alpha_i > 0$), $\epsilon_i = 0$, and therefore $\beta_i > 0$. From (L3), we deduce that $0 < \alpha_i < \frac{1}{n\nu}$. Finally, we define BSV and SV as:

$$
\begin{aligned}
\mathbf{SV} &= \{i \ | \alpha_i > 0, \ \forall i\} \\
\mathbf{BSV} &= \left\{ i \ \middle| 0 < \alpha_i < \frac{1}{n\nu}, \ \forall i \right\}
\end{aligned}
$$

In simple terms, please see figure 27 for better understanding of SV and BSV sets.

$$\mathbf{BSV} \subseteq \mathbf{SV}$$

4.7 Formulating a Data-driven Uncertainty Set

We are going to define an uncertainty set based on the findings in section 4.6. Now, we are looking for vector $a^{(j)}$ in which points are inside or on the boundary of sphere. We refer to the definition of the sphere:

$$R^2 = \|\phi_j - p\|^2 = \|\phi_j - \sum_{i=1}^{n}\alpha_i\phi_i\|^2$$

and by simplifying, we get:

$$R^2 = \mathbf{k}\left(a^{(j)}, a^{(j)}\right) - 2\sum_{i=1}^{n}\alpha_i\mathbf{k}\left(a^{(j)}, a^{(i)}\right) + \sum_{i=1}^{n}\alpha_i^2\mathbf{k}\left(a^{(i)}, a^{(i)}\right)$$

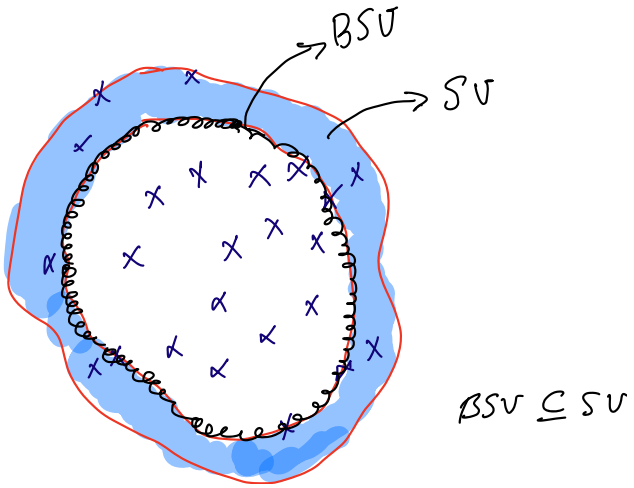

Figure 27: Support Vectors: Illustrated

Now, we define the uncertainty set based on uncertain parameters $a^{(j)}$:

$$\mathcal{U}(a) = \left\{ a^{(j)} \middle| \mathbf{k}\left(a^{(j)}, a^{(j)}\right) - 2\sum_{i\in\mathbf{SV}} \alpha_i \mathbf{k}\left(a^{(j)}, a^{(i)}\right) + \sum_{i\in\mathbf{SV}} \alpha_i^2 \mathbf{k}\left(a^{(i)}, a^{(i)}\right) \leq R^2 \right\}$$

hence,

$$\mathcal{U}(a) = \left\{ a^{(j)} \middle| \mathbf{k}\left(a^{(j)}, a^{(j)}\right) - 2\sum_{i\in\mathbf{SV}} \alpha_i \mathbf{k}\left(a^{(j)}, a^{(i)}\right) + \sum_{i\in\mathbf{SV}} \alpha_i^2 \mathbf{k}\left(a^{(i)}, a^{(i)}\right) \leq \right.$$
$$\left. \mathbf{k}\left(a^{(i')}, a^{(i')}\right) - 2\sum_{i\in\mathbf{SV}} \alpha_i \mathbf{k}\left(a^{(i')}, a^{(i)}\right) + \sum_{i\in\mathbf{SV}} \alpha_i^2 \mathbf{k}\left(a^{(i)}, a^{(i)}\right), \quad \forall i' \in \mathrm{BSV} \right\}$$

Assuming a symmetric kernel, we obtain (we also drop index j):

$$\mathcal{U}(a) = \left\{ a \middle| -2\sum_{i\in\mathbf{SV}} \alpha_i \mathbf{k}\left(a, a^{(i)}\right) \leq -2\sum_{i\in\mathbf{SV}} \alpha_i \mathbf{k}\left(a^{(i')}, a^{(i)}\right), \quad \forall i' \in \mathrm{BSV} \right\} \tag{14}$$

Now, a is a variable and we need to find as that satisfies the inequalities inside the uncertainty set. And consequently, we need to find a decent kernel. Unfortunately, the kernels like Gaussian kernel is not suitable for our case, since it makes the inequality inside the uncertainty set non-linear (or non-convex) and it would be difficult to deal with. Therefore, we are going to use a linear piece-wise kernel (intersection kernel) from the source below:

- Maji, Subhransu, Alexander C. Berg, and Jitendra Malik. "Classification using intersection kernel support vector machines is efficient." In 2008 IEEE conference on computer vision and pattern recognition, pp. 1-8. IEEE, 2008.

$$\textbf{intersection kernel:} \quad \mathbf{k}(r,s) = \sum_{k=1}^{m} l_k - \|Q(r-s)\|_1$$

$$l_k > \max_{1\le i\le n} q_k^T r^{(i)} - \min_{1\le i\le n} q_k^T r^{(i)}, \quad \forall k \in \{1,\cdots,m\}$$

$$Q = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_m \end{bmatrix} = \Sigma^{-\frac{1}{2}}$$

$$\Sigma = \frac{1}{n-1}\left[\sum_{i=1}^{n} r^{(i)}\left(r^{(i)}\right)^T - \left(\sum_{i=1}^{n} r^{(i)}\right)\left(\sum_{i=1}^{n} r^{(i)}\right)^T\right]$$

where Σ is the covariance matrix and Q is commonly referred to the **whitening matrix** to 1) make features less correlated to each other (eliminate cross-correlation) 2) making all features have the same variance (by measuring the similarities of two sample series). Note that the kernel matrix will be $n \times n$ in dimension.

Remark. To obtain the whitening matrix, several steps should be done based on singular value decomposition. You can get the snippet of calculations in python in the following link:

https://github.com/namakshenas/ZCA-whitening-matrix

Next, we plug the this kernel in 14. Then, we obtain:

$$\mathcal{U}(a) = \left\{ a \,\middle|\, \sum_{i\in\mathbf{SV}} \alpha_i \overbrace{\left\|Q\left(a-a^{(i)}\right)\right\|_1}^{w_i \textbf{(dependent on variable } a\textbf{)}} \le \overbrace{\sum_{i\in\mathbf{SV}} \alpha_i \left\|Q\left(a^{(i')}-a^{(i)}\right)\right\|_1}^{\theta \textbf{(fixed and not dependent on } a\textbf{)}} \quad \forall i' \in \text{BSV} \right\}$$

We can also rewrite term $\left\|Q\left(a-a^{(i)}\right)\right\|_1 = w_i$ as an inequality as follows:

$$\left|Q\left(a-a^{(i)}\right)\right| \le w_i, \quad ,\forall i \in \mathbf{SV}$$

or simply:

$$-w_i \le Q\left(a-a^{(i)}\right) \le w_i, \quad ,\forall i \in \mathbf{SV}$$

Hence, the uncertainty set becomes:

$$\mathcal{U}(a) = \left\{ a \;\middle|\; \sum_{i \in \mathbf{SV}} \alpha_i w_i \le \theta, \right.$$
$$\left. -w_i \le Q\left(a - a^{(i)}\right) \le w_i, \quad , \forall i \in \mathbf{SV} \right\}$$

Theorem 4.4. *Uncertainty set $\mathcal{U}(a)$ subject to $0 < \nu < 1$ is bounded and non-empty.*

Remark. If we didn't use a linear kernel, then we couldn't have say that the uncertainty set is convex! And we should've proved it.

Theorem 4.5. *Uncertainty set $\mathcal{U}_\nu(a)$ covers the uncertainty space with $(1-\nu)100\%$ confidence.*

Don't forget our ultimate goal! what is the robust counterpart:

$$\mathbf{P}: \max_{x \in X} \quad c^T x$$
$$\text{s.t.} \quad \max_{\tilde{a} \in \mathcal{U}_\nu(a)} \left\{ \tilde{a}^T x \right\} \le b.$$

We start with the dual of the inner max function (a compact model):

$$\begin{array}{lllllll}
\max\limits_{a, w_i} & 0 & + & \boldsymbol{a^T x} & & & \\
\text{s.t.} & \sum\limits_{i \in \mathbf{SV}} \alpha_i \cdot \mathbf{1}^T \boldsymbol{w_i} & + & 0 & \le & \theta & \eta \ge 0 \text{ dual} \\
& -\boldsymbol{w_i} & + & \boldsymbol{Qa} & \le & \boldsymbol{Qa^{(i)}}, \quad \forall i \in \mathbf{SV} & \boldsymbol{\lambda_i} \ge 0 \text{ dual} \\
& -\boldsymbol{w_i} & - & \boldsymbol{Qa} & \le & -\boldsymbol{Qa^{(i)}}, \quad \forall i \in \mathbf{SV} & \boldsymbol{\mu_i} \ge 0 \text{ dual}
\end{array}$$

Note that nD arrays are in bold format. Remember that we had m parameters and n samples from each. Then, for the above model and the next model, we should have dimensional consistency:

$$\begin{array}{cccc}
\boldsymbol{a}: m \times 1 & \boldsymbol{x}: m \times 1 & \boldsymbol{w_i}: m \times 1 & \mathbf{1}, \mathbf{0}: m \times 1 \\
\boldsymbol{\mu_i}: m \times 1 & \boldsymbol{\lambda_i}: m \times 1 & \boldsymbol{a^{(i)}}: m \times 1 & \boldsymbol{Q}: m \times m \\
\theta: 1 \times 1 & \eta: 1 \times 1 & b: 1 \times 1 & \alpha_i: 1 \times 1
\end{array}$$

Then, the dual is:

$$\min_{\boldsymbol{\lambda_i}, \boldsymbol{\mu_i}, \eta} \quad \sum_{i \in \mathbf{SV}} \left(\boldsymbol{Qa^{(i)}}\right)^T (\boldsymbol{\lambda_i} - \boldsymbol{\mu_i}) + \eta\theta$$
$$\text{s.t.} \quad \boldsymbol{\lambda_i} + \boldsymbol{\mu_i} = \eta \cdot \alpha_i \cdot \mathbf{1}, \quad \forall i \in \mathbf{SV}$$
$$\sum_{i \in \mathbf{SV}} \boldsymbol{Q}\left(\boldsymbol{\lambda_i} - \boldsymbol{\mu_i}\right) - \boldsymbol{x} = \mathbf{0}$$

And the robust counterpart (according to boundedness and non-emptiness in theorems 3.4 and 3.5):

$$\begin{aligned}
\max \quad & c^T x \\
\text{s.t.} \quad & \sum_{i\in\mathbf{SV}} \left(\boldsymbol{Q}\boldsymbol{a}^{(i)}\right)^T (\boldsymbol{\lambda_i} - \boldsymbol{\mu_i}) + \eta\theta \leq b \\
& \boldsymbol{\lambda_i} + \boldsymbol{\mu_i} = \eta \cdot \alpha_i \cdot \mathbf{1}, \quad \forall i \in \mathbf{SV} \\
& \sum_{i\in\mathbf{SV}} \boldsymbol{Q}\left(\boldsymbol{\lambda_i} - \boldsymbol{\mu_i}\right) - \boldsymbol{x} = \mathbf{0} \\
& \boldsymbol{x} \in \mathbb{R}, \boldsymbol{\lambda_i}, \boldsymbol{\mu_i}, \boldsymbol{\eta} \in \mathbb{R}_+
\end{aligned}$$

Note that the final model is compact. And the number of constraints is $1 + \mathbf{card}(\mathbf{SV}) + 1 = 2 + \mathbf{card}(\mathbf{SV})$. So, for each constraint in the original model, we should add the above $2 + \text{card(SV)}$ constraints.

Finally, let's summarize everything in four major step:

1. Calculate the whitening matrix (preprocess).
2. Compute the kernel matrix (preprocess).
3. Solve QP model and construct SV and BSV sets (preprocess).
4. Construct the robust counterpart with SV, BSV, α_i, and Q (modeling).

Example 9. Consider the following simple linear model with two uncertain parameters and two variables:

$$\begin{aligned}
\max \quad & 3x_1 + 2x_2 \\
\text{s.t.} \quad & \tilde{a}_1 x_1 + \tilde{a}_2 x_2 \leq 50, \\
& x_1, x_2 \in \mathbb{R}_+
\end{aligned} \tag{15}$$

Suppose that $\tilde{a}_1 \in [15, 28]$ and $\tilde{a}_2 \in [3, 18]$. In the following, we are going to investigate the robustness of the model based on a box uncertainty set. We call this model **a priori robust model**.

```
# A priori robust optimization model
import numpy as np
from gurobipy import Model,GRB

lp_priori = Model("lp_priori")

a1 = np.arange(15,28)
a2 = np.arange(3,18)
```

```
c = np.array([3,2])

x = lp_priori.addVars(2)

# box uncertainty
lp_priori.addConstr(np.max(a1)*x[0] + np.max(a2)*x[1] <= 50)

lp_priori.setObjective(sum(c[i]*x[i] for i in range(2)),GRB.MAXIMIZE)

lp_priori.optimize()

print("-----------------------------------------------------")
print("objective: ", np.round(lp_priori.objVal,2))
print("x: ", np.round([x[s].X for s in range(2)],2))

# objective:  5.88
# x:  [0.   2.94]
```

Next, we are realizing 100 samples. Hence, its data-driven robust counterpart is,

$$
\begin{aligned}
\max \quad & [3,2]^T \cdot \boldsymbol{x} \\
\text{s.t.} \quad & \sum_{i \in \mathbf{SV}} \left(\boldsymbol{Q}\boldsymbol{a}^{(i)}\right)^T (\boldsymbol{\lambda}_i - \boldsymbol{\mu}_i) + \eta\theta \leq 50 \\
& \boldsymbol{\lambda}_i + \boldsymbol{\mu}_i = \eta \cdot \alpha_i \cdot \mathbf{1}, \quad \forall i \in \mathbf{SV} \\
& \sum_{i \in \mathbf{SV}} \boldsymbol{Q}\left(\boldsymbol{\lambda}_i - \boldsymbol{\mu}_i\right) - \boldsymbol{x} = \mathbf{0} \\
& \boldsymbol{x} \in \mathbb{R}_+, \boldsymbol{\lambda}_i, \boldsymbol{\mu}_i, \boldsymbol{\eta} \in \mathbb{R}_+
\end{aligned}
$$

```
import numpy as np
import pandas as pd
from numpy import linalg as LA

# loading data

# a = np.array([[.65,5.03,2.72,1.93,2.71],
#       [.79,7.44,5.37,4.12,5.29]])

# mean = [20, 10]
# cov = [[1, 1.5], [1.5, 3]]  # mxm
# a = np.random.multivariate_normal(mean, cov, 100).T # mxn

# pd.DataFrame(a).to_csv("C:/Users/Monash/Documents
```

```
# /sample-linear-lp-ddro.csv",index=False)


import os
os.chdir('C:/Users/Monash/Documents/')
os.getcwd()
df = pd.read_csv("sample-linear-lp-ddro.csv")
a = df.to_numpy() # 2x100


# visualizing data

import matplotlib.pyplot as plt

fig, axes = plt.subplots()

# a should be 100x2
axes.violinplot(a.T ,showmeans=True, showmedians=True, showextrema=True)

plt.plot(1, np.min(a1), 'o' ,color='red')
plt.plot(1, np.max(a1), 'o' ,color='red')

plt.plot(2, np.min(a2), 'o' ,color='red')
plt.plot(2, np.max(a2), 'o' ,color='red')

plt.xticks([1,2])

plt.grid()
fig.set_size_inches(18.5, 10.5)

plt.show()
```

The following is a plot containing a priori supports and the realized data:

```
def whitening_matrix(X):
        sigma = np.cov(X, rowvar=True) # [M x M]
        U,S,V = LA.svd(sigma)
        epsilon = 1e-5
        ZCAMatrix = np.dot(U,
         np.dot(np.diag(1.0/np.sqrt(S + epsilon)), U.T)) # [M x M]
        return ZCAMatrix

Q = whitening_matrix(a)
```

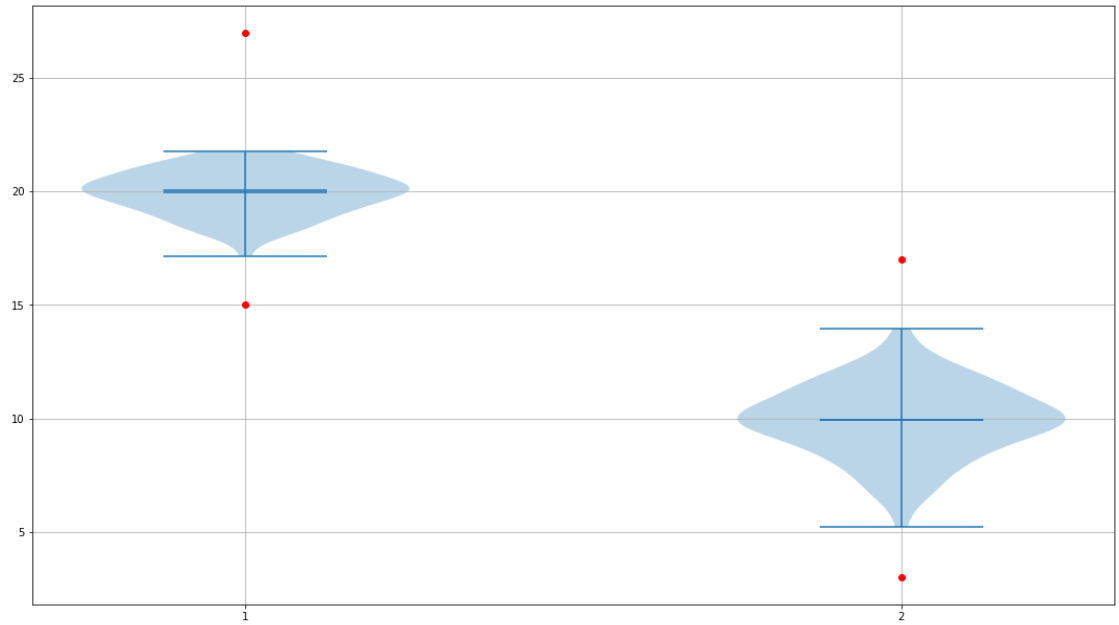

Figure 28: Violin plot of a priori and the realized data

```
l_psd = np.zeros(a.shape[0])
for i in range(a.shape[0]):
        l_ub = np.max(np.dot(Q[i,:].T,a))
        l_lb = np.min(np.dot(Q[i,:].T,a))
        l_psd[i] = l_ub - l_lb + .0001

ker_mat = np.zeros((a.shape[1],a.shape[1]))
for u in range(a.shape[1]):
        for v in range(a.shape[1]):
        ker_mat[u,v] = np.sum(l_psd) -
        LA.norm(np.dot(Q, a[:,u] - a[:,v]),1)


#checking PSD
np.linalg.eigvals(ker_mat)
```

Then, we are going to model and solve the QP model to obtain SV and BSV:

```
from gurobipy import Model,GRB

svc = Model("support-vector-clustering")
```

```
p_regul = .1 # nu
n_samples = a.shape[1] # n
p_ker_mat = ker_mat
v_alpha = svc.addVars(a.shape[1], lb = 0, ub = 1/(a.shape[1]*p_regul))

svc.addConstr(sum(v_alpha[i] for i in range(n_samples))==1)

svc.setObjective( sum(v_alpha[i]*v_alpha[j]*p_ker_mat[i,j]
for i in range(n_samples)
for j in range(n_samples)) -
sum(v_alpha[i]*p_ker_mat[i,i] for i in range(n_samples)) )

svc.optimize()

# getting alpha and rounding
v_alpha_dict = [v_alpha[s].X for s in range(a.shape[1])]
v_alpha_array_trunc = np.round(np.array(v_alpha_dict),2)

# lets print alphas
print(v_alpha_array_trunc)

# lets build BSV and SV
SV = np.where(v_alpha_array_trunc > 0)
BSV = np.where((v_alpha_array_trunc > 0) &
(v_alpha_array_trunc < 1/(a.shape[1]*p_regul)) )

SV = np.array(SV).ravel()
BSV = np.array(BSV).ravel()
print(SV)
print(BSV)
#[ 7  9 26 31 34 41 42 44 68 75 79 84 96]
#[ 7 26 41 42 75 79 84]

# lets visualize
import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()
plt.plot(a[0,:], a[1,:], 'x',color='blue',label="interior")
plt.plot(a[0,SV], a[1,SV], 'o',color='red',label="SV")
plt.plot(a[0,BSV], a[1,BSV], 'o',color='black',label="BSV")
plt.axis('square')
```

```
plt.legend()
fig.set_size_inches(18.5, 10.5)
plt.show()
```

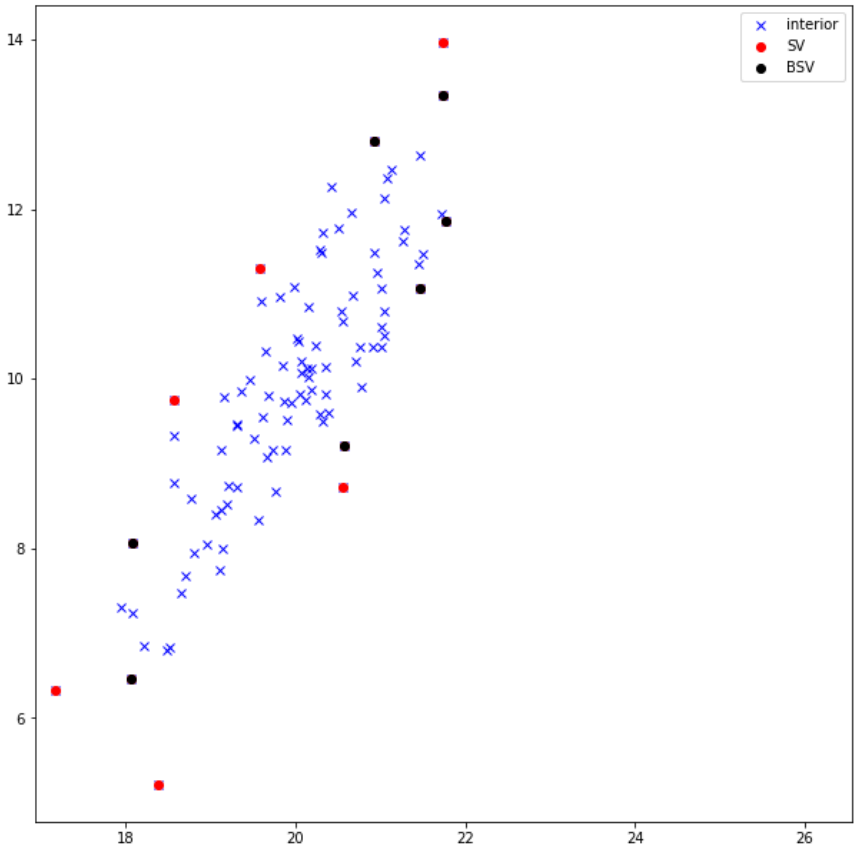

Figure 29: Illustration of support vectors in example 9

Next, we are going to build the robust counterpart:

```
p_alpha_SV = v_alpha_array_trunc[SV]
p_a_SV = a[:,SV]
p_a_BSV = a[:,BSV]

from gurobipy import Model,GRB
ddro = Model("data-driven-robust-opt-svc")

p_num_SV = SV.size
print(p_num_SV)

p_rhs = 50
```

```
p_obj_coef = np.array([3,2])
p_num_BSV = BSV.size
p_a_BSV

# here we choose minimum BSV since it produces smaller
# theta and tight uncertainty set
p_theta_vec = np.zeros(p_num_BSV)
for ip in range(p_num_BSV):
p_theta_vec[ip] = np.sum([p_alpha_SV[i]*
LA.norm(np.dot(Q,p_a_BSV[:,ip]-p_a_SV[:,i]),1)
for i in range(p_num_SV)])

p_theta = np.min(p_theta_vec)
print(p_theta)


v_dual_eta = ddro.addVar()
v_primal_x = ddro.addMVar(2)
v_dual_la = ddro.addMVar((2,p_num_SV))
v_dual_mu = ddro.addMVar((2,p_num_SV))

ddro.addConstr(sum(np.dot(Q,p_a_SV[:,i]).T @ v_dual_la[:,i]
for i in range(p_num_SV)) -
sum(np.dot(Q,p_a_SV[:,i]).T @ v_dual_mu[:,i]
for i in range(p_num_SV)) + v_dual_eta*p_theta <= p_rhs)

for j in range(2):
for i in range(p_num_SV):
ddro.addConstr(v_dual_la[j,i] + v_dual_mu[j,i] == v_dual_eta*p_alpha_SV[i])

ddro.addConstr(sum( Q @ v_dual_la[:,i] for i in range(p_num_SV)) -
sum( Q @ v_dual_mu[:,i] for i in range(p_num_SV)) - v_primal_x == 0)


ddro.setObjective(p_obj_coef @ v_primal_x,GRB.MAXIMIZE)


ddro.optimize()
status = ddro.status
if status == GRB.OPTIMAL:
print("-------------------------------------------------")
print("objective: ", np.round(ddro.objVal,2))
```

```
print("x: ", np.round([v_primal_x[s].X for s in range(2)],2))

#-----------------------------------------------------------
#objective:  7.55
#x:  [0.   3.78]
```

4.8 Data abstraction and Lazy updating

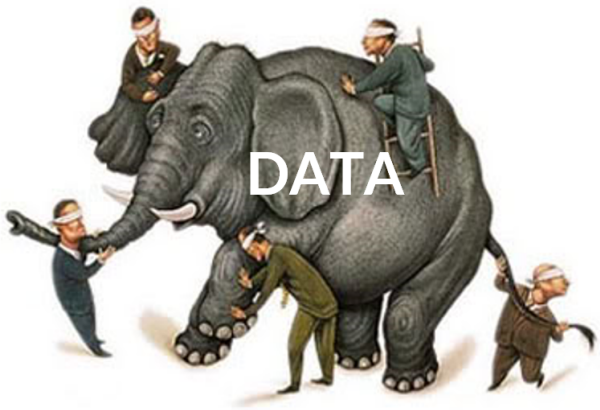

Figure 30: Idea- Rumi's Elephant in the Room

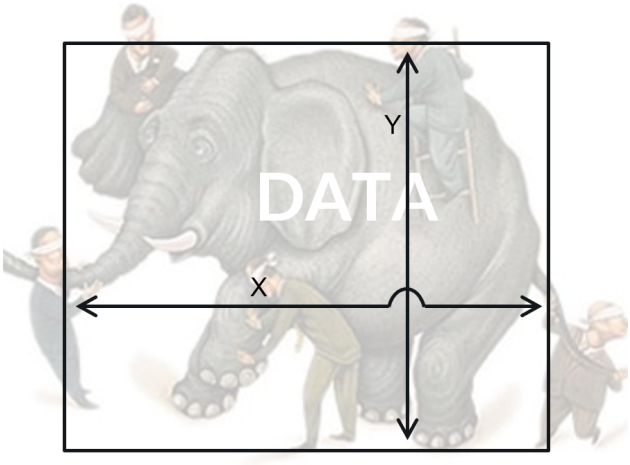

Figure 31: Idea- Data sketch example

Suppose we have support of s_0 in the information base and new data is realized as s_1 such that $|s_1| < |s_0|$, $s_0^+ > s_1^+$, $s_1^- > s_0^-$ where $s_0 = [s_0^-, s_0^+]$ and $s_1 = [s_1^-, s_1^+]$. I haven't seen a serious

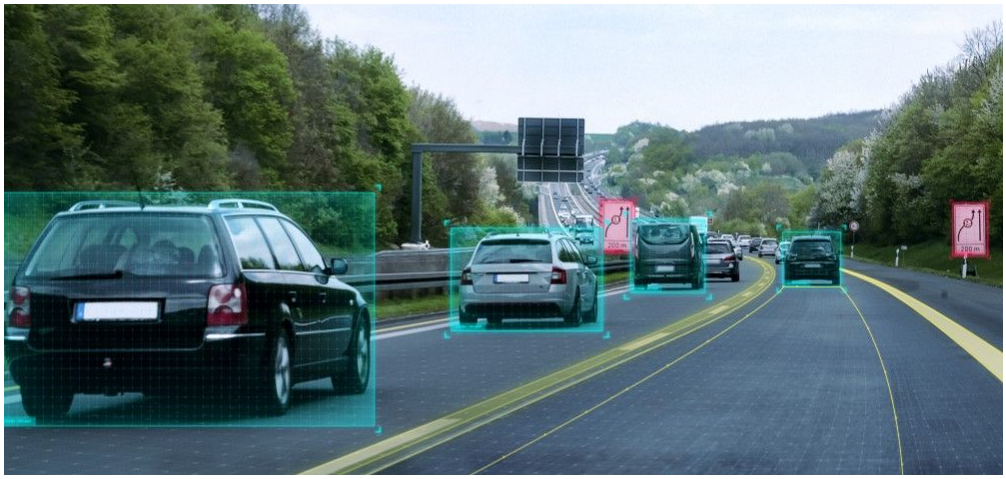

Figure 32: Idea- Data sketch practical example

work on this. From this point on-wards, you can think of future discussion. Hence, two types of strategies can be proposed (similar to Bayesian updating in Statistics or Recourse function in Adjustable Robust Optimization):

- prompt updating
- lazy updating (risk-averse updating)

Lazy updating is more interesting, since it can smartly choose a support which is a [convex or not] combination of current (k) and the past (k-1) realization. For example, a lazy updating model can be formulated as a damped sine function:

$$\max_{x^-,x^+} \quad \left(s_0^+ - s_0^-\right) e^{x^-/w^- + x^+/w^+} \left(\cos \frac{s_1^- - x^-}{s_1^- - s_0^-}\pi + \sin \frac{x^+ - s_1^+}{s_0^+ - s_1^+}\pi\right)$$

where w^+ and w^- are the weights for the importance of current observations. In other words, term $e^{x^-/w^- + x^+/w^+} \left(\cos \frac{s_1^- - x^-}{s_1^- - s_0^-}\pi + \sin \frac{x^+ - s_1^+}{s_0^+ - s_1^+}\pi\right)$ is oscillating between $[0, 1]$ and the whole term is oscillating between $[s_0^-, s_0^+]$. Therefore, variables x^-, x^+ shows how far we tend to incorporate the past realizations.

The uncertainty set which is derived from this a posteriori support is called a posteriori uncertainty set ($\mathcal{U}_{post}(a)$).

Now, we are going to use a powerful optimization package `scipy` to solve the damped function (a simple example):

```
# a damped sine function to update the support
def damped_fun(x):
```

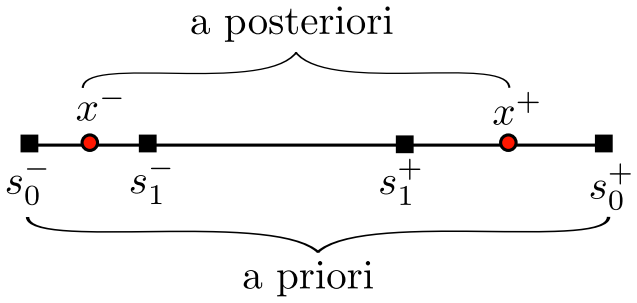

Figure 33: Idea: how to derive a posteriori support

```
x_minus = x[0]
x_plus = x[1]
damped_sin_fun = (s_0_plus-s_0_minus)*np.exp(-1*(x_minus/w_minus +
 x_plus/w_plus))*(np.cos(np.pi*(s_1_minus-x_minus)/(s_1_minus-s_0_minus))+
  np.sin(np.pi*(x_plus-s_1_plus)/(s_0_plus-s_1_plus)))
return -damped_sin_fun

import numpy as np
from scipy.optimize import dual_annealing

# for a1; then run the below function
s_0_plus = np.max(a1)
s_0_minus = np.min(a1)
s_1_plus = np.max(a[0,:])
s_1_minus = np.min(a[0,:])
w_minus = .1
w_plus = .2

# so we can play with weights!

lw = [s_0_minus,s_1_plus]
up = [s_1_minus,s_0_plus]
ret = dual_annealing(damped_fun, bounds=list(zip(lw, up)))

a1_post = np.round(ret.x,2)
print(a1_post)



# for a2; then run the below function again
s_0_plus = np.max(a2)
```

```
s_0_minus = np.min(a2)
s_1_plus = np.max(a[1,:])
s_1_minus = np.min(a[1,:])
w_minus = 1
w_plus = 2

lw = [s_0_minus,s_1_plus]
up = [s_1_minus,s_0_plus]
ret = dual_annealing(damped_fun, bounds=list(zip(lw, up)))

a2_post = np.round(ret.x,2)
print(a2_post)


# now generate new parameter samples
a_post_a1 = np.hstack([a[0,:],np.linspace(a1_post[0],a1_post[1],num=10)])
a_post_a2 = np.hstack([a[1,:],np.linspace(a2_post[0],a2_post[1],num=10)])
a = np.vstack([a_post_a1,a_post_a2])
print(a)


import matplotlib.pyplot as plt

fig, axes = plt.subplots()

# a should be 100x2
axes.violinplot(a.T ,showmeans=True, showmedians=True, showextrema=True)

plt.plot(1, np.min(a1), 'o' ,color='red')
plt.plot(1, np.max(a1), 'o' ,color='red')

plt.plot(2, np.min(a2), 'o' ,color='red')
plt.plot(2, np.max(a2), 'o' ,color='red')

plt.plot(1, a1_post[0], 'o' ,color='black')
plt.plot(1, a1_post[1], 'o' ,color='black')

plt.plot(2, a2_post[0], 'o' ,color='black')
plt.plot(2, a2_post[1], 'o' ,color='black')

plt.xticks([1,2])
```

```
plt.grid()
fig.set_size_inches(13, 7)

plt.show()
```

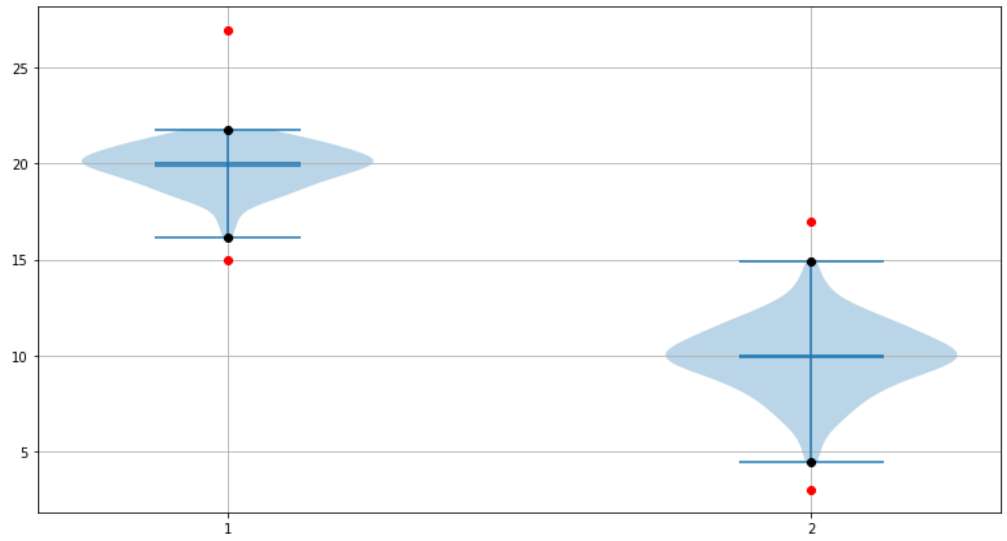

Figure 34: Lazy updating for example 9

Example 10. The knapsack problem is a problem in combinatorial optimization: Given a set of items, each with a weight (w) and a utility (u), determine which item should be selected (x) to include in a collection so that the total weight ($\sum_{j=1}^{n} w_j x_j$) is less than or equal to a given limit (C) and the total utility ($\sum_{j=1}^{n} u_j x_j$) is as large as possible. Weights $\tilde{w}_j$ are not deterministic and their respective stochastic states are oscillating within $\tilde{w}_j : [w_j - \hat{w}_j, w_j + \hat{w}_j]$. Note that we have only one constraint!

$$
\begin{aligned}
\max \quad & \boldsymbol{u}^T \boldsymbol{x} \\
\text{s.t.} \quad & \sum_{i \in \mathbf{SV}} \left(\boldsymbol{Q} \boldsymbol{w}^{(i)} \right)^T (\boldsymbol{\lambda_i} - \boldsymbol{\mu_i}) + \eta\theta \leq C, \\
& \boldsymbol{\lambda_i} + \boldsymbol{\mu_i} = \eta \cdot \alpha_i \cdot \mathbf{1}, \quad \forall i \in \mathbf{SV} \\
& \sum_{i \in \mathbf{SV}} \boldsymbol{Q} \left(\boldsymbol{\lambda_i} - \boldsymbol{\mu_i} \right) - \boldsymbol{x} = \mathbf{0} \\
& \boldsymbol{\lambda_i}, \boldsymbol{\mu_i}, \boldsymbol{\eta} \in \mathbb{R}_+ \\
& \boldsymbol{x} \in \{0, 1\}.
\end{aligned}
$$

Example 11. supply chain problem ($j = 1, \cdots, q,\ k = 1, \cdots, m$)

$$\begin{aligned}
\textbf{minimize} \quad & \sum_k f_k y_k + \sum_k \sum_j \tilde{c}_{kj} x_{kj} \\
\text{subject to} \quad & \sum_j x_{kj} \le u_k y_k,\ \forall k, \\
& \sum_k x_{kj} \ge d_j,\ \forall j, \\
& y_k \in \{0,1\}, \quad x_{kj} \ge 0,\ \forall k, j.
\end{aligned}$$

and its matrix format:

$$\begin{aligned}
\textbf{minimize} \quad & z \\
\text{subject to} \quad & f^T y + \textbf{trace}\left(\tilde{c}^T x\right) \le z, \\
& \textbf{diag}\,(x\mathbf{1}) \le u^T y, \\
& \textbf{diag}\left(x^T \mathbf{1}\right) \ge \textbf{diag}(d), \\
& y \in \{0,1\}, \quad x \in \mathbb{R}_+.
\end{aligned}$$

worst case:

$$\begin{aligned}
\textbf{minimize} \quad & z \\
\text{subject to} \quad & f^T y + \max_{\tilde{c} \in \mathcal{U}} \left\{ \textbf{trace}\left(\tilde{c}^T x\right) \right\} \le z, \\
& \textbf{diag}\,(x\mathbf{1}) \le u^T y, \\
& \textbf{diag}\left(x^T \mathbf{1}\right) \ge \textbf{diag}(d), \\
& y \in \{0,1\}, \quad x \in \mathbb{R}_+.
\end{aligned}$$

ddro counter part

$$\begin{aligned}
\textbf{minimize} \quad & z \\
\text{subject to} \quad & f^T y + \sum_{i \in \mathbf{SV}} \left(\boldsymbol{Q c^{(i)}}\right)^T + (\boldsymbol{\lambda_i} - \boldsymbol{\mu_i}) + \eta \le z, \\
& \textbf{diag}\,(x\mathbf{1}) \le u^T y, \\
& \textbf{diag}\left(x^T \mathbf{1}\right) \ge \textbf{diag}(d), \\
& \boldsymbol{\lambda_i} + \boldsymbol{\mu_i} = \eta \cdot \alpha_i \cdot \mathbf{1}, \quad \forall i \in \mathbf{SV}, \\
& \sum_{i \in \mathbf{SV}} \boldsymbol{Q}\,(\boldsymbol{\lambda_i} - \boldsymbol{\mu_i}) - \boldsymbol{x} = \mathbf{0}, \\
& y \in \{0,1\}, \quad \textbf{diag}\left(x^T \mathbf{1}\right) \in \mathbb{R}_+.
\end{aligned}$$

A punchline

Benefints of using SVC in RO enjoys is an adaptive complexity, thereby featuring an non parametric scheme. Most importantly, it leads to a convex polyhedral uncertainty set, thereby rendering the robust counterpart problem of the same type as the deterministic problem, which provides computational convenience. If the deterministic problem is an MILP, then the robust counterpart problem can be also cast as an MILP.

References

[Ber09] Bertsekas, D. P., "Convex Optimization Theory," *Belmont: Athena Scientific*, 2009.

[Ben09] Ben-Tal, A. , El Ghaoui, L., and Nemirovski, A. "Robust Optimization," *Princeton University Press*, 2009.

[Her15] Hertog, D. D., "Practical Robust Optimization: An Introduction," *Lecture Notes*, 2015.

[Spr12] Spreij, P. JC., "Measure Theoretic Probability," *UvA Course Notes*, 2012.

[Ben13] Benson, H. Y.,Sağlam, U., "Mixed-integer second-order cone programming: A survey," *Theory Driven by Influential Applications* , 2013, pp. (13-36). INFORMS.