

Practical Session 4, 2019-2020 (B)

1. Aim of the Session

In this session, you will learn how to find the limit of functions, the derivative of functions and the integral functions using SymPy. You will also learn how to find the global minimum and maximum using SymPy. In addition, you will learn more data manipulation with Pandas.

2. Preparation

- Click the **window icon** on the bottom left corner:
- Search **Anaconda Navigator** by typing **Anaconda** from the search box under All Programs.
- Click on **Anaconda 3.6 Navigator**
- Click **Launch** under Jupyter from Anaconda Navigator;
- Once the jupyter notebook is open, what you can see are files and/or folders under C:\Docs
- Click **New** from Jupyter, then select **Python 3** from the pop-up list. You should see a new page with a blank cell, where you can type in Python code.
- Click File and select Rename from the pop-up list, you may give the file a name you prefer, for example, practical4.

3. Solving Calculus Problems

In this section, you will learn how to solve basic calculus problems using SymPy. We assume that all functions in this session you are calculating the derivative of are differentiable in their respective domains.

- Finding the limit of functions
Consider a function $f(x) = 1/x$.

Task 1:

Generate a graph of $f(x)$ when x is in the range of [10, 1000] with xlabel and ylabel.

You can find limits of functions in SymPy by creating objects of the **Limit** class as follows:

```
from sympy import Limit, Symbol, S
x = Symbol('x')
l=Limit(1/x, x, S.Infinity)
l
Limit(1/x, x, oo, dir='-')
```

In the above code, the first line imports the **Limit**, **Symbol**, and **S** classes. **S** contains the definition of some special values, such as positive infinity, negative infinity, and so on. In the next line, a symbol object, x , is created. The Limit object, l , is created with three arguments: the function expression, the variable and the value at which you want to calculate the function's limit. The result is returned with the ∞ symbol denoting positive infinity and the $\text{dir}='-'$ symbol denoting that the limit is approached from the negative side.

To find the value of the limit, you can use the **doit()** method:

```
l=Limit(1/x, x, S.Infinity)
l.doit()
```

0

Task2:

Calculate $\lim_{x \rightarrow 0} \frac{1}{x}$

- Finding the derivations of functions

The derivative of a function $y = f(x)$ is the rate of change in the dependent variable y with respect to the independent variable x . It is denoted as either $f'(x)$ or $\frac{dy}{dx}$.

Consider a function $f(x) = 100x^2 + 4x + 9$. You can find the derivative of this function by creating an object of the **Derivative** class as follows:

```
from sympy import Symbol, Derivative
```

```
x = Symbol('x')
f = 100*x**2 + 4*x + 9
Derivative(f, x)
```

```
Derivative(100*x**2 + 4*x + 9, x)
```

First import the Symbol class and the Derivative class. Then define the independent variable and the function expression. An object of the Derivative class is created by passing with two arguments: the function and the variable. You can obtain the derivative by calling the **doit()** method:

```
df = Derivative(f, x).doit()
```

```
df
```

```
200*x + 4
```

You may calculate the value of the derivative at a particular value of x , say $x = 1$. You can use the **subs()** method:

```
df.subs({x:1})
```

```
204
```

Task3:

Consider a function $f(x) = x^3 + 2x^2 + \ln x$

- Find the derivative of $f(x)$ by yourself and write down your answer on a paper.
 - Find the derivative of $f(x)$ in Python using SymPy and compare the result with your answer. Note that to call the **ln()** function, **import sympy** first, then use **sympy.ln()** function.
 - Calculate the value of the derivative at $x = 2$ using the **subs()** method.
- Calculating partial derivatives
For partial differentiation, we assume only one variable varies while the others are fixed. Consider the function $f(m, n) = m^2n + 10mn$, the partial differentiation of $f(m, n)$ with respect to m is:

```
m = Symbol('m')
n = Symbol('n')
f = m**2*n + 10*m*n
df = Derivative(f,m)
df.doit()
```

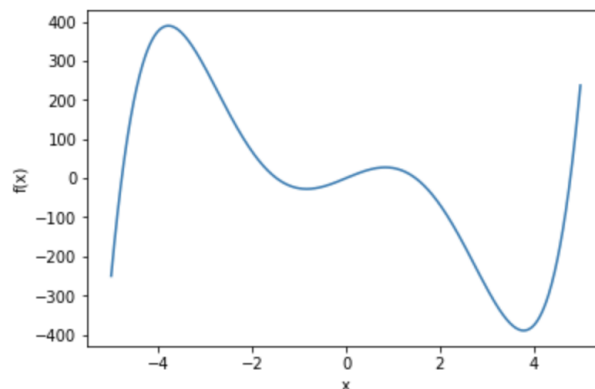
2*m*n + 10*n

Task4:

Calculate the partial differentiation of $f(m, n)$ with respect to n . First write down your answer on a paper. Then compare the result obtained from Python with your answer.

- Higher-order derivatives and finding the maxima and minima

Consider the function $f(x) = x^5 - 25x^3 + 50x$, and the following graph shows the function when x is in the range of $[-5, 5]$. As one can see, the function has local minimum, local maximum, global minimum and global maximum values. Recall that the first-order derivative of a function is zero at the local or global maximum or minimum, that is $f'(x) = 0$. Solutions of $f'(x) = 0$ are called critical points of the function.



Task5:

Calculate the first-order derivative of $f(x) = x^5 - 25x^3 + 50x$ using SymPy and save the result in the variable say, **dy**

Now you can solve $f'(x) = 0$ and find the critical points by doing the following:

```
from sympy import solve

critical_points = solve(dy)
critical_points

[-sqrt(-sqrt(185)/2 + 15/2),
 sqrt(-sqrt(185)/2 + 15/2),
 -sqrt(sqrt(185)/2 + 15/2),
 sqrt(sqrt(185)/2 + 15/2)]
```

To find out the global maximum or minimum, you need to calculate the second derivative of the function. To do so, you can enter 2 as the third argument of

Derivative():

```
dy2 = Derivative(y, x, 2).doit()
dy2

10*x*(2*x**2 - 15)
```

You can find the value of the second derivative by substituting the value of each critical point in place of x . If the resulting value is less than zero, the point is a local

maximum; if the value is greater than zero, it is a local minimum. If the resulting value is zero, then we cannot deduce anything about whether the critical point x is a local minimum, maximum, or not.

```
dy2.subs({x:critical_points[0]}).evalf()  
113.738286404346
```

```
dy2.subs({x:critical_points[1]}).evalf()  
-113.738286404346
```

```
dy2.subs({x:critical_points[2]}).evalf()  
-514.357465393284
```

```
dy2.subs({x:critical_points[3]}).evalf()  
514.357465393284
```

It can be seen that critical points 1 and 4 correspond to the local minimum; critical points 2 and 3 corresponding to the local maximum.

The global maximum and minimum of $f(x)$ in the interval $[-5, 5]$ is attained either at a critical point or at one of the endpoints of the domain ($x = -5$ and $x = 5$). To find the global minimum, you must compute the value of $f(x)$ at the critical point 1, critical point 4, $x = -5$ and $x = 5$. Among these points, the place where $f(x)$ has the largest value must be the global minimum.

```
x_min=-5  
x_max=5  
y.subs({x:critical_points[0]}).evalf()  
-27.6014252624441
```

```
y.subs({x:critical_points[3]}).evalf()  
-389.535828035730
```

```
y.subs({x:x_min}).evalf()  
-250.000000000000
```

```
y.subs({x:x_max}).evalf()  
250.000000000000
```

As can be seen, the global minimum of $f(x)$ is about -389.54 at critical point 4.

Now please spend a couple of minutes in thinking carefully about how the minimum and maximum values of a function have been found. Can you remember the procedure? If not, you need to go back and have a look at this section again. Otherwise, you may continue.

- Finding the integrals of functions

Consider to find the integral $\int sxdx$. You can find the integral by creating an object of the **Integral** class first as follows:

```
from sympy import Integral, Symbol

s = Symbol('s')
x = Symbol('x')
Integral(s*x, x)

Integral(s*x, x)
```

Similar to **Limit** and **Derivative** classes, you can now evaluate the integral using the **doit()** method:

```
Integral(s*x, x).doit()

s*x**2/2
```

To find the definite integral, for example, $\int_5^0 sxdx$, you can simply specify the variable with the lower and upper limits:

```
from sympy import Integral, Symbol

s = Symbol('s')
x = Symbol('x')
Integral(s*x, (x, 0, 5)).doit()
```

Task6:

Calculate the integrate $\int_6^2 x dx$

- 1) Do it by yourself and write your answer on a paper
- 2) Finish it by using SymPy and compare with your result.

4. Gradient Ascent Algorithm

Download demo_gradient_ascent.ipynb from Canvas. Open it from your jupyter notebook. This is a generic program for gradient ascent in a simple version.

Task7:

- 1) Enter function $x^5 - 30 \times x^3 + 50 \times x$, and the variable to differentiate with respect to is x .
- 2) Observe how the maximum value changes when you enter different initial values. For example, you may enter the first initial value -2, and then 0.5.

5. Datasets

During learning on this module, you need to apply what you have learnt on various datasets in order to understand and memorise processes. This section shows you how and where you can obtain some datasets for you to use.

- Scikit-learn toy datasets

Scikit-learn (<http://scikit-learn.org/stable>) is the core of data science operations on Python. You will learn more about it later in this module. The Scikit-learn toy datasets are included in the Scikit-learn package. You can load these datasets directly into Python by the import command as follows:

```
from sklearn import datasets

iris = datasets.load_iris()
```

where iris is one of the toy datasets. To see which datasets have been included and the details about each dataset, you can view here: <https://scikit-learn.org/stable/datasets/index.html>.

All Scikit-learn datasets provide the following methods:

- .DESCR: a description of the data. For example: iris.DESCR
- .data: This shows all the data. For example: iris.data
- .feature_names: This shows the names of the features. For example: iris.feature_names
- .target: This contains the target values expressed as values or numbered classes. For example: iris.target
- .target_names: This shows the names of the classes in the target. For example: iris.target_names
- .shape: this is a method that you can apply to both .data and .target. For example: iris.data.shape

Task8:

Load the Boston dataset from the Scikit-learn toy datasets. Understand the dataset by using the methods in the above list.

- UCI Machine Learning Repository

You can also download datasets from the UCI webpage (<https://archive.ics.uci.edu/ml/index.php>)

6. Data Manipulation with Pandas

The pandas package (<http://pandas.pydata.org>) deals with everything that NumPy cannot do. It allows you to handle complex tables of data of different types and time series [2]. You can import pandas and check its version as follows:

```
import pandas as pd
pd.__version__
```

- Fast and easy data loading

The most common format for machine learning data files is CSV files. There are a number of ways to load a CSV file in Python

(<https://machinelearningmastery.com/load-machine-learning-data-python/>).

Let's start with opening a CSV using pandas. For example:

```
import pandas
url = "http://aima.cs.berkeley.edu/data/iris.csv"
names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'target']
iris_data = pandas.read_csv(url, sep=',', decimal='.', header=None, names=names)
```

As can be seen, you can specify the name of the file, the character used as a separator, the character used for the decimal placeholder (decimal), the header and the variables names using a list.

To get a rough idea of this data file's content, you can print the first or the last rows using the following commands:

	sepal_length	sepal_width	petal_length	petal_width	target
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

	sepal_length	sepal_width	petal_length	petal_width	target
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

Task9:

Type the following commands line by line and observe what you get from each line:

```
iris_data['sepal_length']
iris_data['target']
iris_data.head
print(iris_data.shape)
```

As you can see, the last column ‘target’ is in the string format. If you want to convert it to numerical labels, you may try the following:

```
labels = pd.factorize(iris_data.target)[0]
```

labels

[illegible]

- Convert a Scikit-learn dataset to a Pandas dataset

To use more functions from Pandas to deal with data, sometimes you may want to convert a Scikit-learn dataset to a Pandas dataset. For example;

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

newdata = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                        columns= iris['feature_names'] + ['target'])
```

where `np.c_` is the concatenate function from the numpy module. `np.c_` concatenates `iris['data']` and `iris['target']` arrays. For Pandas column argument concatenates the `iris['feature_names']` list and the string list `['target']`.

And you can type the following to check the data type:

```
newdata.dtypes

sepal length (cm)    float64
sepal width (cm)     float64
petal length (cm)    float64
petal width (cm)     float64
target              float64
dtype: object
```

Further, if you want to change the data type, for example, you may want to change 'target' from float64 to the integer type (Int64), you can type the following:

```
newdata['target'] = newdata['target'].astype('Int64')
```

Now if you type `newdata.dtypes`, you should get something similar to the following:

```
newdata.dtypes

sepal length (cm)    float64
sepal width (cm)     float64
petal length (cm)    float64
petal width (cm)     float64
target              Int64
dtype: object
```

- Dealing with problematic data

In practice, it is very common to have messy data including missing values and dates. You need to download `a_messy_example.csv` from Canvas and open it using pandas as follows:

```
import pandas

messy_data = pandas.read_csv('a_messy_example.csv', sep=',')
messy_data
```

	Date	Temperature_city_1	Temperature_city_2	Destination
0	20171210	5.0	20	1
1	20171211	2.0	23	1
2	20171212	3.0	19	2
3	20171213	8.0	21	1
4	20171214	5.0	20	2
5	20171215	NaN	21	1
6	20171216	3.0	18	2

As can be seen, all the data, even the dates, has been parsed as integers. You may try the following code to convert dates into the proper format.

```
messy_data = pandas.read_csv('a_messy_example.csv', sep=',', parse_dates=[0])
```


In addition, to get rid of the missing data that is indicated as NaN, you may either fill in a proper value, or drop off the row including the missing values. For example,

	Date	Temperature_city_1	Temperature_city_2	Destination
0	2017-12-10	5.0	20	1
1	2017-12-11	2.0	23	1
2	2017-12-12	3.0	19	2
3	2017-12-13	8.0	21	1
4	2017-12-14	5.0	20	2
5	2017-12-15	4.0	21	1
6	2017-12-16	3.0	18	2

Task10:

Drop off the row from the messy data using the method.dropna() and check the resulting data.

Note that another possible problem when handling real world datasets is the loading of a dataset with errors or bad lines. A possible workaround, which is not always feasible, is to ignore this line [2]. For example, suppose that you have a badly formatted data file called a_messy_data2.csv. What you may try to do is as follows:

```
bad_data = pandas.read_csv('a_messy_data2.csv', error_bad_lines=False)
```

- Data selection

- Removing the index column

As you may have noticed, the first column of data which are read into Python is index. This index column should not be used by mistake as a feature. If you need to remove it, you may do the following:

```
messy_data = pandas.read_csv('a_messy_example.csv', sep=',', index_col=0)
messy_data
```

	Temperature_city_1	Temperature_city_2	Destination
Date			
20171210	5.0	20	1
20171211	2.0	23	1
20171212	3.0	19	2
20171213	8.0	21	1
20171214	5.0	20	2
20171215	NaN	21	1
20171216	3.0	18	2

Task11:

Please remove the index column when load the Iris data using pandas.

- Access the value of a cell

You can simply specify the column and the line you are interested in. Note that it is not a matrix. This is actually a Panda's DataFrame, and the [] operator works first on columns and then on the element of the resulting pandas Series.

```
iris_data['petal_width'][10]
```

0.2

If you want to access the data using something similar to access matrices, you may use the `.loc()` method.

```
iris_data.loc[10, 'petal_width']  
0.2
```

Task12:

Access the fifth value under the column of 'sepal_length' using the both methods described above.

- Access values of sub-matrixes

You may access values of sub-matrixes as follows:

```
iris_data[['sepal_length', 'petal_width']][0:10]
```

Or

```
iris_data.loc[range(0,10), ['sepal_length', 'petal_width']]
```

- Access values using `.loc()` with conditions

For example, if you want to find all patterns with a value of **sepal_width** less than 5cm, you may type the following:

```
iris_data.loc[iris_data.sepal_width<5]
```

	sepal_length	sepal_width	petal_length	petal_width	target
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

.
. .
.

Furthermore, type the following to see what you get:

```
iris_data.loc[(iris_data.sepal_width<5) & (iris_data.sepal_width>4) ]
```

References

1. Amit Saha: Doing math with Python: use programming to explore algebra, statistics, calculus and more!
2. Alberto Boschetti and Luca Massaron: Python data science essentials: become an efficient data science practitioner by thoroughly understanding the key concepts of Python. 2015 Packt Publishing