```
#--------------------------
# INTEGRALS and DERIVATIVES
#--------------------------
# Chapter 5 of the book


# Some YouTube videos about integrals
# https://www.youtube.com/watch?v=eis11j_iGMs
# https://www.youtube.com/watch?v=4grhQ5Y_MWo

# SciPy page about integrals
# https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html


# SINGLE INTEGRAL: area under a curve
# DOUBLE INTEGRAL: volume under a surface


#--------------------------
#         SCIPY
#--------------------------
# We can do derivatives and integrals using SCIPY
#
# Page with explanantions about SciPy
# https://www.southampton.ac.uk/~fangohr/teaching/python/book/html/16-
 scipy.html
#
# Numpy module: vectors, matrices and linear algebra tools.
# Matplotlib package: plots and visualisations.
# Scipy package: a multitude of numerical algorithms.
#
# Many of the numerical algorithms available through scipy and numpy are
# provided by established compiled libraries which are often written
# in Fortran or C. They will thus execute much faster than pure Python code.
# As a rule of thumb, we expect compiled code to be two orders of magnitude
# faster than pure Python code.


# Scipy is built on numpy.
# All functionality from numpy seems to be available in scipy as well.

import numpy as np
x = np.arange(0, 10, 1.)
y = np.sin(x)
print(y)

import scipy as sp
x = sp.arange(0, 10, 1.)
y = sp.sin(x)
print(y)
```

```
#%%

#--------------------------
#   LAMBDA FUNCTION
#--------------------------
print()
# Lambda function is similar to a definition,
# but
# *) the lambda definition does not include a return statement;
# *) we can put a lambda definition anywhere a function is expected;
# *) we don't have to assign it to a variable at all.

# Go through:
# https://www.southampton.ac.uk/~fangohr/teaching/python/book/html/16-
 scipy.html
# https://www.tutorialspoint.com/scipy/scipy_integrate.htm
#
# Compare
def func(x):
    return x ** 3
print(func(5))


print()
fanc = lambda x: x ** 3
print(fanc(5))



#%%

#
#     NUMERICAL INTEGRAL



# ------------
# Example 1
# -----------
print()
import scipy.integrate as integrate
import numpy as np
import math

func = lambda x: math.exp(-x**2)
# ------------------------------
# numpy could also have been used
#func = lambda x: np.exp(-x**2)
# ------------------------------------
```

```
# definition could also have been used
#def func(x): return math.exp(-x**2)

sol1=integrate.quad(func, 0, 1)
print(sol1)
# The solution contains two numbers.
# The first is the solution of the of integral
# and the second value is
# the estimate of the absolute error in the value of the integral.
#
# If we just want to see the solution, we can write
print()
print("Printing just the solution:")
sol,err=integrate.quad(func, 0, 1)
print(sol)

# The scipy documentation states that integrate.quad uses
# a technique from the Fortran library QUADPACK.
# QUADPACK provides several routines using different techniques.
# As scipy is open source, you can actually read the code for integrate.quad at
# https://github.com/scipy/scipy/blob/master/scipy/integrate/quadpack.py



#%%

#
#
#        MULTIPLE INTEGRALS
#
#
#
# quad: Single integration
# dblquad: Double integration
# tplquad: Triple integration
#
#
# https://www.tutorialspoint.com/scipy/scipy_integrate.htm

# ------------
# Example 2
# -----------
import scipy.integrate
from numpy import exp
from math import sqrt

func = lambda x, y : 16*x*y
lowY = lambda x : 0
highY = lambda x : sqrt(1-4*x**2)
int = scipy.integrate.dblquad(func, 0, 0.5, lowY, highY)
print (int)
```

```
# %%

# MORE EXAMPLES   MORE EXAMPLES    MORE EXAMPLES

# ---------------------
#   SINGLE INTEGRAL
# ---------------------
import scipy.integrate as integrate
import numpy as np
import math

# Either we define the function like this
def func(x):
    return x ** 3
sol=integrate.quad(func, 0, 1)
print(sol)

# Or we lambdify it
fanc = lambda x: x ** 3
sol=integrate.quad(fanc, 0, 1)
print(sol)


# ---------------------
#   DOUBLE INTEGRAL
# ---------------------
from scipy import integrate
import numpy as np
import math

# EITHER
def func(y,x):
    return 1/math.sqrt(x*y-x**2)
def flow(x):
    return x+2
def fup(x):
    return x+3
sol=integrate.dblquad(func, 1, 2, flow, fup)
print(sol)
# VERY CAREFUL above!! The order of the variables in func matters!!!
# If we are going to integrate y first and then x, make sure to have
#                func(y,x) and NOT func(x,y)


# OR
func = lambda y, x: 1/math.sqrt(x*y-x**2)
sol=integrate.dblquad(func, 1, 2, lambda x: x+2, lambda x: x+3)
print(sol)
```

```
# OR yet
def func(y,x):
    return 1/math.sqrt(x*y-x**2)
sol=integrate.dblquad(func, 1, 2, lambda x: x+2, lambda x: x+3)
print(sol)



#%%

#---------------------------
#         SYMPY
#---------------------------

# https://www.sympy.org/en/index.html

# SumPy is a Python library for symbolic mathematics.
#
#     SYMBOLIC INTEGRAL



# ------------
# Example 3
# -----------
import scipy.integrate as integrate
import sympy as syp
x = syp.Symbol('x')
intSym = syp.integrate(3.0*x**2+1,x)
print(  intSym   )




#%%

#
#     METHODS of INTEGRATION
#
# Trapezoidal rule
# Simpson's rule
# Romberg integration
# Gaussian quadrature


# ------------------------------------
# Example 4:
# Integral of a semicircle of radius 1
# ------------------------------------
print()
```

```
import scipy.integrate as integrate
import numpy as np

func = lambda x: np.sqrt(1.-x**2)
print()
print("The exact solution is Pi/2:",np.pi/2.)

print()
print("Using quad from scipy")
sol,err=integrate.quad(func, -1, 1)
print("Solution:",sol, " Error:", err)

# --------------
# RECTANGLES
#---------------

print()
print("Writing our own code: adding rectangles")

print()
print("Just 10 intervals")
Ntot=10
som=0.
Xlast=1.
Xfirst=-1.
delt=(Xlast-Xfirst)/Ntot
print("increment=",delt)
for n in range(Ntot):
    print("where func is evaluated:",Xfirst+n*delt)
    som=som + func(Xfirst+n*delt)*delt
print("Solution for 10 intervals:",som)

print()
Ntot=100
print(Ntot, "intervals")
som=0.
Xlast=1.
Xfirst=-1.
delt=(Xlast-Xfirst)/Ntot
for n in range(Ntot):
    som=som + func(Xfirst+n*delt)*delt
print("Solution:",som)

print()
Ntot=1000
print(Ntot, "intervals")
som=0.
Xlast=1.
Xfirst=-1.
delt=(Xlast-Xfirst)/Ntot
for n in range(Ntot):
```

```
    som=som + func(Xfirst+n*delt)*delt
print("Solution:",som)




#%%
#
#
# ----------------------------------------
# STUDENTS READ THE BOOK (see Canvas)
# about the TRAPEZOIDAL RULE and SIMPSON'S RULE

#
# STUDENTS PRESENT ON THE BOARD
# (student selected by chance)
# One student presents the TRAPEZOIDAL RULE
# One student presents the SIMPSON'S RULE
# This exercise can be done in pairs or groups
#
#
#
# ----------------------------------------
# LET US NOW CONSIDER THE FUNCTION
#
#        x^4 - 2 x +1
# in the interval x=0 to x=2
#
# and compare the results that we obtain
# by adding rectangles for N=10,100,100 and
# by employing
# the TRAPEZOIDAL RULE and SIMPSON'S RULE
# ----------------------------------------


func = lambda x: x**4 - 2.*x + 1.
Xfirst = 0
Xlast = 2.

print()
print("Exact solution")
var=2
exac=var**5/5 - var**2 + var
print("x^5/5-x^2+x in [0,2] is ", exac)

print()
Ntot=10
print(Ntot, "intervals")
delt=(Xlast-Xfirst)/Ntot
som=0.
print("increment=",delt)
```

```
for n in range(Ntot):
    som=som + func(Xfirst+n*delt)*delt
print("Solution:",som)
print("Percentage Error:", 100*abs(som-exac)/exac,"%")

print()
Ntot=100
print(Ntot, "intervals")
delt=(Xlast-Xfirst)/Ntot
som=0.
for n in range(Ntot):
    som=som + func(Xfirst+n*delt)*delt
print("Solution:",som)
print("Percentage Error:", 100*abs(som-exac)/exac,"%")

print()
Ntot=1000
print(Ntot, "intervals")
delt=(Xlast-Xfirst)/Ntot
som=0.
for n in range(Ntot):
    som=som + func(Xfirst+n*delt)*delt
print("Solution:",som)
print("Percentage Error:", 100*abs(som-exac)/exac,"%")



# ------------------
# TRAPEZOIDAL RULE
#------------------
# Exercise 1
# Students solve in class for Ntot=10,100,1000


# ----------------------------------------
# SIMPSON'S RULE
# The number of slices need to be EVEN !!!
#----------------------------------------
# Exercise 2
# Students solve in class for Ntot=10,100,1000



# ------------------------------------------
# ROMBERG INTEGRATION
# and
# GAUSSIAN QUADRATURE
# Read the book if you want to learn about them.
#------------------------------------------
```

```
#%%
# ------------------------------------------------

# --------------------
# ERRORS ON INTEGRALS
#--------------------

# When the integrand f(x) is known,
# there are equations available for

# 1) the error using the TRAPEZOIDAL RULE is
#        error = [f'(a)-f'(b)]*(h^2/12)
#
# 2) the error using the SIMPSON'S RULE is
#        error = [f'''(a)-f''''(b)]*(h^4/90)
#
# where
# f' = first derivative
# f''' = third derivative
# Integral performes from x=a to x=b
# h = (b-a)/N and N = number of slices


# ------------------------------
# PRACTICAL ESTIMATION OF ERRORS
# ------------------------------
# valid also when f(x) is not known
#
# Solve the integral with N1 steps and get I1
# Solve the integral with N2=2*N1 steps and get I2

# 1) the error for I2 using the TRAPEZOIDAL RULE is
#        error = abs(I2 - I1)/3
#
# 2) the error for I2 using the SIMPSON'S RULE is
#        error = abs(I2 - I1)/15


# -----------
# Exercise 3
# -----------
# Students solve in class.
# Use the trapezoidal rule with 20 slices and
# calculate the integral of x^4 - 2 x +1
# in the interval x=0 to x=2.
# Using the "practical estimation of errors",
# compute also the error. For this, get the integral for N1=10 and N2=20.
```

```
# ------------------------------
# CHOOSING THE NUMBER OF STEPS
# that gives a desired accuracy
# ------------------------------
# Read the Section 5.3 of the book
# You will need this to solve Exercise 4 of the assignment



#%%

# ------------------------------
# INTEGRALS OVER INFINITE RANGES
# ------------------------------
# The previous methods do not work,
# because we would need an infinite number of sample points.
# The solution to this problem is to change variables.
#
# For an integral in x from 0 to infinity,
# the standard change of variables from x to z is
#
# z = x/(1+x)   so   x = z/(1-z)
#
# Then dx = dz/(1-z)^2
# and
# \int_0^infinity f(x) dx = \int_0^1 ( 1/(1-z)^2 ) f(z/(1-z)) dz
#
# Many other choices work, so sometimes we need to play around
# to find which works better, but
# the choice above is often a first good guess (see exception in assignment!).

# When the range is from a to infinity,
# change x to y:    y = x-a
# and then y to z:  z = y/(1+y)
# See equations in the book.

# When the range is from -infinity to a,
# change z to -z

# When the range is from -infinity to infinity,
# split the integrals.


# -----------
# Exercise 4
# -----------
# Students solve in class the integral
# \int_0^infinity exp(-t^2) dt
#
# !!!!!!! CAREFUL !!!!!!!!!!
# At z=1, 1/(1-z) goes to infinity,
```

```
# so use z = 0.999999999999999


# To solve it analytically, look at YouTube movie
# https://www.youtube.com/watch?v=nqNzKeVCYBU
#
# Show in class that:
# \int_{-infinity}^{infinity} exp(-a x^2) dx = sqrt(pi/a)
#
# Use
# (i)  \int_{-infinity}^{infinity} exp(-a x^2) dx =
# sqrt[ (\int_{-infinity}^{infinity} exp(-a x^2) dx) (\int_{-infinity}
 ^{infinity} exp(-a y^2) dy) ]

# (ii) x = r cos(theta)  and   y = r sin(theta)
# Jacobian: | dx/dr  dx/dtheta |
#           | dy/dr  dy/dtheta | = r dr dtheta

# (iii) z = r^2   and dz = 2 r dr



# %%

# ------------
# DERIVATIVES
# ------------

# Forward difference
# f' ~ [ f(x+h) - f(x) ]/h for small h

# Backward difference
# f' ~ [ f(x) - f(x-h) ]/h for small h

# Central difference [BETTER OPTION]
# f' ~ [ f(x+h/2) - f(x-h/2) ]/h for small h

# For second derivatives and partial derivatives
# see the book

# ------------
# Using SYMPY
# ------------

import sympy as syp
x = syp.Symbol('x')
ff = syp.diff(syp.sin(x),x)
print(ff)
```