

數值分析Numerical Analysis

許志宇Chin-Yu Hsu
October 1, 2018

聽眾Audience

- 碩一first year graduate
- 基礎課程 Basic curriculum
- 19 students

Objective

- 引導您了解如何自學。Guide you to know how to learn by yourself.
- 數值分析（NA）的歷史是什麼？
- What is the history of the Numerical Analysis (NA)?
 - Who are the important people NA? Newton, Lagrange, Gauss and Euler.
 - What are the important events of NA?
 - Where is the location of NA?
 - Which objects are related to NA?
 - When is NA popular?

材料Materials

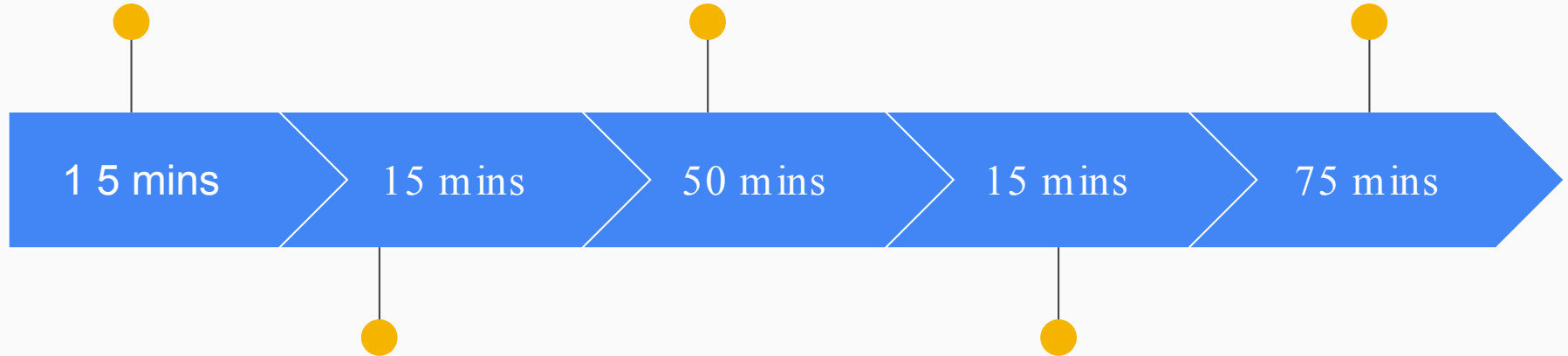
- What is NA
- Python Language
- Tensor flow
- Project and report

程序Procedure

Introduction

Step by step to learn
small concepts by
doing

A project and report
should be done by
yourself



Demo:How to solve
problem?

Conclusions

家庭作業1/Homework1

Learning efficiency depends on motivation

1. Why do you choose this course?
2. What kind of jobs for NA?
3. What kind of jobs you want?

How to learn knowledge of NA efficiently?

1. Python programming
2. Mathematics
3. Tensorflow, scikit learn and, keras

Python programming

Learning topics

1. Using Python as a Calculator
 - a. Arithmetic operations
2. Variables
3. if Statements
4. for Statements
5. The range() Function
6. Executing modules as scripts
7. Mathematics
8. Python Functions - W3Schools

Learning Resources

1. [Welcome to Python.org](https://www.python.org/)
2. [Download](#)
3. [Python For Beginners](#)
4. [The Python Tutorial](#)
5. [IntroductoryBooks](#)
6. [Python Functions - W3Schools](#)
7. [A Visual Introduction to Python](#)
8. [Try Jupyter](#)

Python programming

Learning topics

1. Python Functions

```
def my_function():  
    print("Hello from a function")
```

```
my_function()
```

Learning Resources

1. [Python Functions - W3Schools](#)
2. [Run web python or Anaconda](#)
3. [http://jupyter.org/try](#)
4. [https://hub.mybinder.org/user/jupyterlab-jupyterlab-demo-yqivt6ba/lab#Integration-\(scipy.integrate\)](#)

Python programming

← → ↻ [https://hub.mybinder.org/user/jupyterlab-jupyterlab-demo-yqivt6ba/lab#Integration-\(scipy.integrate\)](https://hub.mybinder.org/user/jupyterlab-jupyterlab-demo-yqivt6ba/lab#Integration-(scipy.integrate)) ☆ 已暫停

File Edit View Run Kernel Tabs Settings Help

demo

Name	Last Modified
data	16 days ago
notebooks	16 days ago
TCGA_Data	16 days ago
Lorenz.ipynb	an hour ago
test_integration.ipynb	12 minutes ago
big.csv	16 days ago
jupyterlab-slides.pdf	16 days ago
jupyterlab.md	16 days ago
lorenz.py	16 days ago
markdown_python...	16 days ago

Lorenz.ipynb × test_integration.ipynb ×

No Kernel

The computation of integral

We calculate the Integral of function $f(x) = x$ in interval $(0,1)$:

$$I = \int_0^1 f(x)dx$$

Let's calculate the value of I .

Learning Latex: Integrals, sums and limits

https://www.overleaf.com/learn/latex/Integrals,_sums_and_lim

We explore the definite integration of function $f(x) = x$:

$$I = \int_0^1 f(x)dx$$
$$= \frac{1}{2}x^2 \Big|_0^1 = \frac{1}{2}1^2 - 0 = \frac{1}{2}$$

1. <http://jupyter.org/try>

Python programming

Plot graph curve $y=f(x)=x$

```
import numpy as np
```

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

Create data

```
x=np.arange(0,1,0.1) # interval from  
0 to 1 with unit 0.1
```

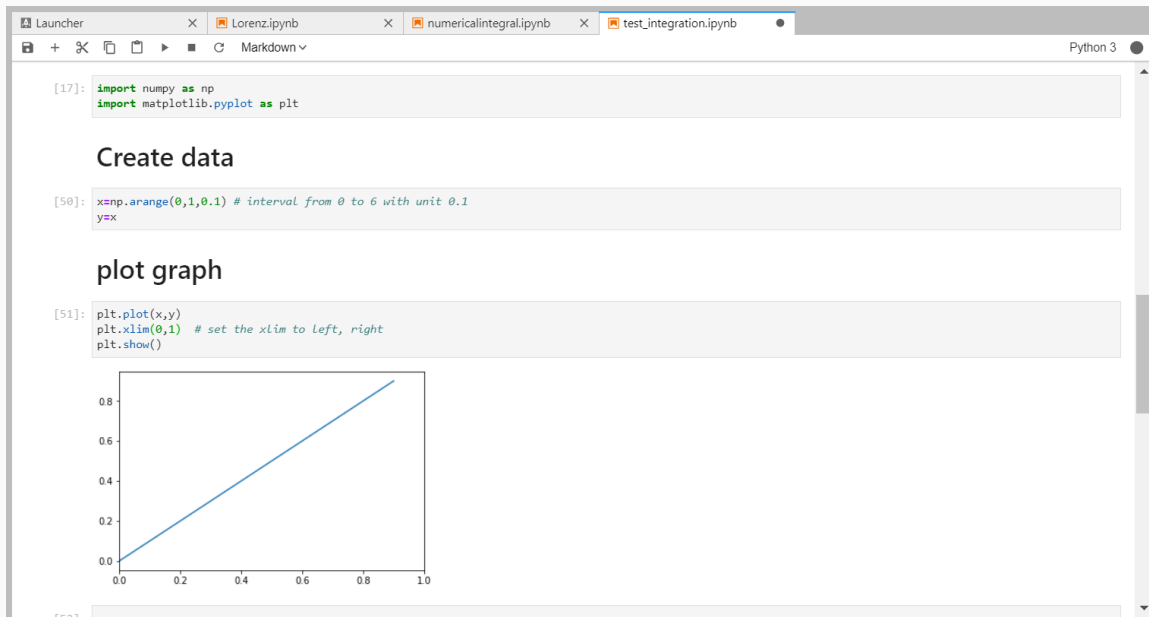
```
y=x
```

plot graph

```
plt.plot(x,y)
```

```
plt.xlim(0,1) # set the xlim to left,  
right
```

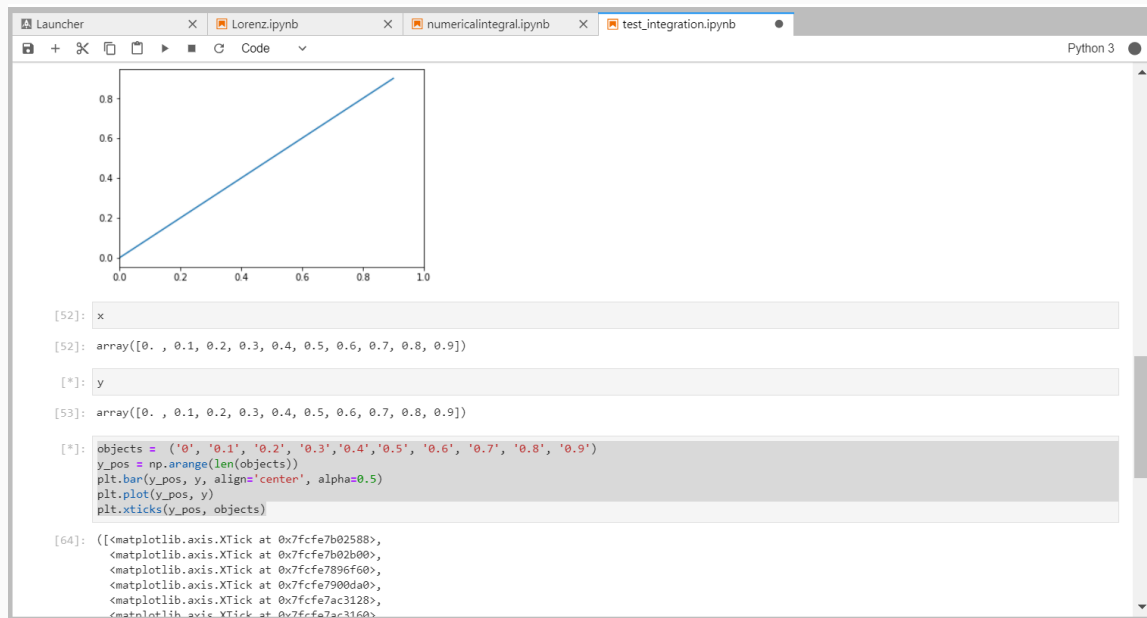
```
plt.show()
```



程式設計 Python programming

Data
X
y

1. <http://jupyter.org/try>



程式設計 Python programming

Bar Plot

Riemann integral

sum of areas of bars (width multiply height)

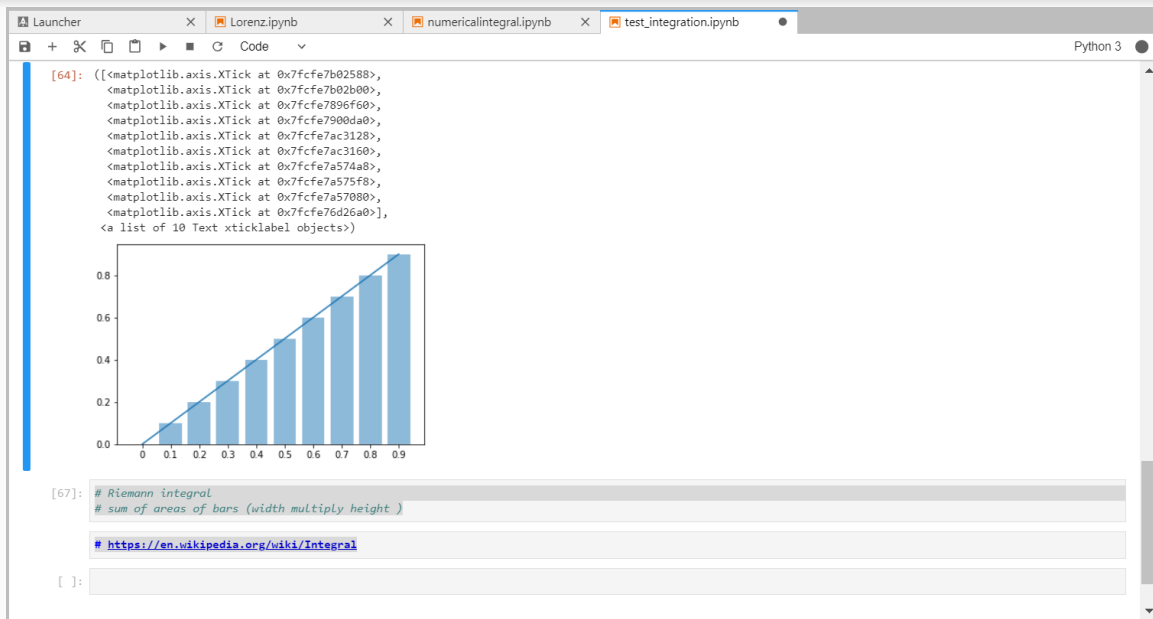
<https://en.wikipedia.org/wiki/Integral>

**objects = ('0', '0.1', '0.2',
'0.3', '0.4', '0.5', '0.6', '0.7', '0.8', '0.9')**

y_pos = np.arange(len(objects))
plt.bar(y_pos, y, align='center',
alpha=0.5)

plt.plot(y_pos, y)
plt.xticks(y_pos, objects)

<http://jupyter.org/try>

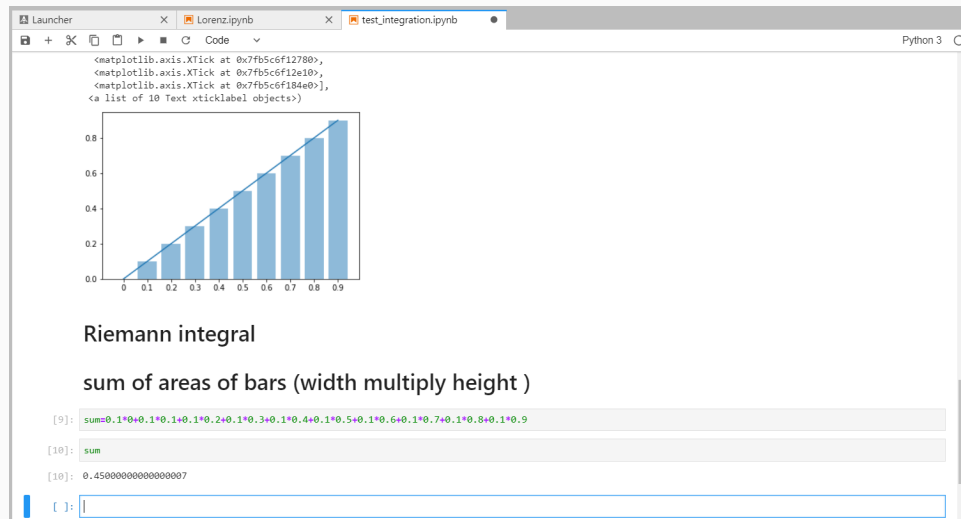


Python programming

Riemann integral

sum of areas of bars (width multiply height)

<https://en.wikipedia.org/wiki/Integral>



https://en.wikipedia.org/wiki/Numerical_integration

Python programming

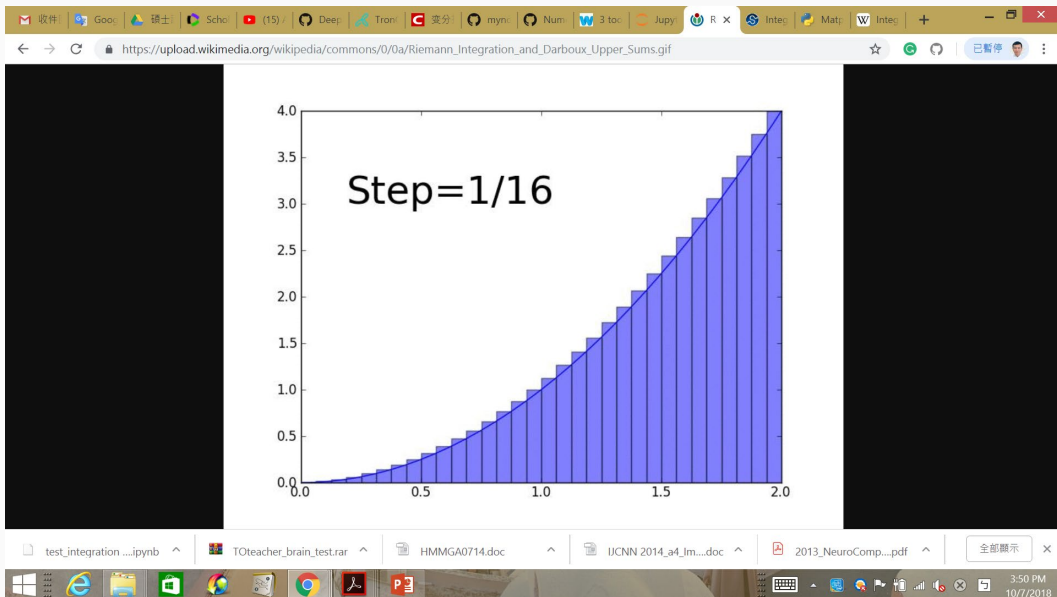
Question:

$$Y=x^2 \quad [0 \ 1]$$

Summing the areas of these rectangles, we get a better approximation for the sought integral, namely

<https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.quad.html>

<https://en.wikipedia.org/wiki/Integral>



Latex programming

Learning to Latex to Integrals, sums and limits

→ ↻ 🏠 🔒 https://www.overleaf.com/learn/latex/Integrals_sums_and_limits ☆ 🟢 🔄 🔍 📄 📁

Creating your first LaTeX document

Choosing a LaTeX Compiler

Paragraphs and new lines

Bold, italics and underlining

Lists

Errors

Mathematics

Mathematical expressions

Subscripts and superscripts

Brackets and Parentheses

Fractions and Binomials

Aligning Equations

Operators

Spacing in math mode

Integrals, sums and limits

Integrals

Integral expression can be added using the

$$\backslash\mathrm{int}_{\mathrm{lower}}^{\mathrm{upper}}$$

command.

Note, that integral expression may seems a little different in *inline* and *display* math mode - in *inline* mode the integral symbol and the limits are compressed.

LaTeX code	Output
Integral $\backslash\mathrm{int}_{\mathrm{a}}^{\mathrm{b}} x^2 dx$ inside text	Integral $\int_a^b x^2 dx$ inside text
$\backslash\mathrm{int}_{\mathrm{a}}^{\mathrm{b}} x^2 dx$	$\int_a^b x^2 dx$

https://www.overleaf.com/learn/latex/Integrals_sums_and_limits

Python programming

Learning to implement the algorithm on wiki

https://en.wikipedia.org/wiki/Riemann_sum

https://en.wikipedia.org/wiki/Riemann_sum

Methods [\[edit\]](#)

The four methods of Riemann summation are usually best approached with partitions of equal size. The interval $[a, b]$ is therefore divided into n subintervals, each of length

$$\Delta x = \frac{b - a}{n}.$$

The points in the partition will then be

$$a, a + \Delta x, a + 2 \Delta x, \dots, a + (n - 2) \Delta x, a + (n - 1) \Delta x, b.$$

Left Riemann sum [\[edit\]](#)

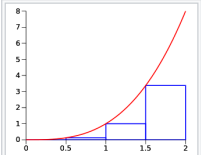
For the left Riemann sum, approximating the function by its value at the left-end point gives multiple rectangles with base Δx and height $f(a + i\Delta x)$. Doing this for $i = 0, 1, \dots, n - 1$, and adding up the resulting areas gives

$$\Delta x [f(a) + f(a + \Delta x) + f(a + 2 \Delta x) + \dots + f(b - \Delta x)].$$

The left Riemann sum amounts to an overestimation if f is [monotonically decreasing](#) on this interval, and an underestimation if it is [monotonically increasing](#).

Right Riemann sum [\[edit\]](#)

f is here approximated by the value at the right endpoint. This gives multiple rectangles with base Δx and height $f(a + i \Delta x)$. Doing this for $i = 1, \dots, n$, and adding up the resulting areas produces

$$\Delta x [f(a + \Delta x) + f(a + 2 \Delta x) + \dots + f(b)].$$


Left Riemann sum of x^3 over $[0, 2]$ using 4 subdivisions

Python programming

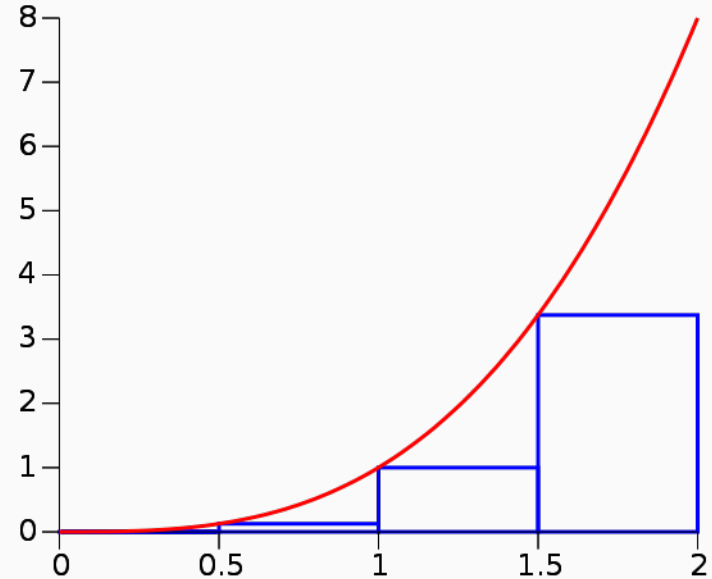
Question:

Refer to:

https://github.com/tccnchsu/Numerical_Analysis/blob/master/NA_W4_Integration.ipynb

$F(x)=x^2$

https://en.wikipedia.org/wiki/Riemann_sum



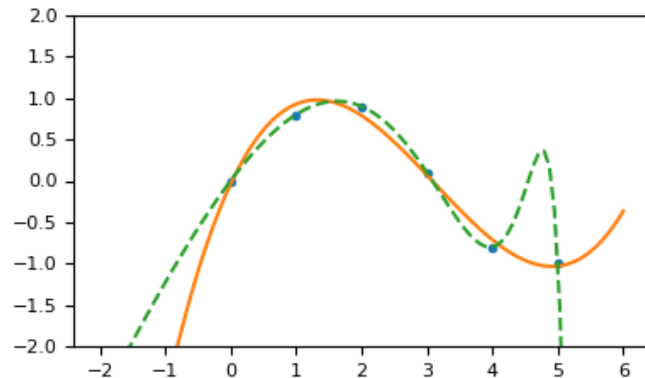
Python programming

polynomial interpolation

Two points \leftrightarrow 2,3,4 degree polynomial

Three points \leftrightarrow 2,3,4 degree polynomial

Four points \leftrightarrow 2,3,4 degree polynomial



<https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.polyfit.html>

Python programming

We will see two types of global (interpolatory) rules:

- Newton-Cotes — interpolatory on uniformly spaced nodes.
- Gauss rules — interpolatory on optimally chosen point sets.

Numerical Integration
Numerical Differentiation
Richardson Extrapolation

Quadrature Rules
Adaptive Quadrature
Other Integration Problems

Quadrature Rules, continued

- Quadrature rules are based on polynomial interpolation
- Integrand function f is sampled at finite set of points
- Polynomial interpolating those points is determined
- Integral of interpolant is taken as estimate for integral of original function
- In practice, interpolating polynomial is not determined explicitly but used to determine weights corresponding to nodes
- If Lagrange interpolation is used, then weights are given by

$$w_i = \int_a^b \ell_i(x), \quad i = 1, \dots, n$$

I

https://relate.cs.illinois.edu/course/cs450-f17/file-version/f753d2d1a50ecbf2ca66202aa894e6d67c24efdb/lecture-notes/chap08_lec2.pdf

Python programming

5-point Gaussian quadrature rule

$$\begin{aligned}\int_a^b f(x) dx &= \int_{-1}^1 f\left(\frac{b-a}{2}t + \frac{a+b}{2}\right) \frac{b-a}{2} dt \\ &\approx \sum_{i=1}^5 f\left(\frac{b-a}{2}r_{5,i} + \frac{a+b}{2}\right) \frac{b-a}{2} c_{5,i} \\ &\approx \sum_{i=1}^5 f(x_i) w_i,\end{aligned}$$

where

$$x_i = \frac{b-a}{2}r_{5,i} + \frac{a+b}{2}, \quad w_i = \frac{b-a}{2}c_{5,i}, \quad i = 1, \dots, 5.$$

In this example, $a = 0$ and $b = 1$, so the nodes and weights for a **5-point Gaussian quadrature rule** for integrating over $[0, 1]$ are given by

$$x_i = \frac{1}{2}r_{5,i} + \frac{1}{2}, \quad w_i = \frac{1}{2}c_{5,i}, \quad i = 1, \dots, 5,$$

which yields

i	Nodes x_i	Weights w_i
1	0.95308992295	0.11846344250
2	0.76923465505	0.23931433525
3	0.50000000000	0.28444444444
4	0.23076534495	0.23931433525
5	0.04691007705	0.11846344250

It follows that

$$\begin{aligned}\int_0^1 e^{-x^2} dx &\approx \sum_{i=1}^5 e^{-x_i^2} w_i \\ &\approx 0.11846344250e^{-0.95308992295^2} + 0.23931433525e^{-0.76923465505^2} + \\ &\quad 0.28444444444e^{-0.5^2} + 0.23931433525e^{-0.23076534495^2} + \\ &\quad 0.11846344250e^{-0.04691007705^2} \\ &\approx 0.74682412673352.\end{aligned}$$

Python programming

Gaussian Quadrature Weights and Abscissae

Weights and Abscissae Tables for $n=2$ to $n=64$

$n = 2$

jump to $n = 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64$

i	weight - w_i	abscissa - x_i
1	1.0000000000000000	-0.5773502691896257
2	1.0000000000000000	0.5773502691896257

$n = 3$

jump to $n = 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64$

i	weight - w_i	abscissa - x_i
1	0.8888888888888888	0.0000000000000000
2	0.5555555555555556	-0.7745966692414834
3	0.5555555555555556	0.7745966692414834

$n = 4$

jump to $n = 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64$

i	weight - w_i	abscissa - x_i
1	0.6521451548625461	-0.3399810435848563
2	0.6521451548625461	0.3399810435848563
3	0.3478548451374538	-0.8611363115940526
4	0.3478548451374538	0.8611363115940526

<https://pomax.github.io/bezierinfo/legendre-gauss.html>

Python programming

Quadrature rules based on interpolating functions

https://en.wikipedia.org/wiki/Numerical_integration

← → ↺ https://en.wikipedia.org/wiki/Numerical_integration ☆ 🗨 已暫停

Quadrature rules based on interpolating functions [\[edit\]](#)

A large class of quadrature rules can be derived by constructing [interpolating](#) functions that are easy to integrate. Typically these interpolating functions are [polynomials](#). In practice, since polynomials of very high degree tend to oscillate wildly, only polynomials of low degree are used, typically linear and quadratic.

The simplest method of this type is to let the interpolating function be a constant function (a polynomial of degree zero) that passes through the point $\left(\frac{a+b}{2}, f\left(\frac{a+b}{2}\right)\right)$. This is called the *midpoint rule* or *rectangle rule*

$$\int_a^b f(x) dx \approx (b-a)f\left(\frac{a+b}{2}\right).$$

The interpolating function may be a straight line (an *affine function*, i.e. a polynomial of degree 1) passing through the points $(a, f(a))$ and $(b, f(b))$. This is called the *trapezoidal rule*

$$\int_a^b f(x) dx \approx (b-a)\left(\frac{f(a)+f(b)}{2}\right).$$

For either one of these rules, we can make a more accurate approximation by breaking up the interval $[a, b]$ into some number n of subintervals, computing an approximation for each subinterval, then adding up all the results. This is called a *composite rule*, *extended rule*, or *iterated rule*. For example, the composite trapezoidal rule can be stated as

$$\int_a^b f(x) dx \approx \frac{b-a}{n} \left(\frac{f(a)}{2} + \sum_{k=1}^{n-1} \left(f\left(a + k \frac{b-a}{n}\right) \right) + \frac{f(b)}{2} \right),$$

where the subintervals have the form $[a + kh, a + (k+1)h] \subset [a, b]$, with $h = \frac{b-a}{n}$ and $k = 0, \dots, n-1$. Here we used subintervals of the same length h but one could also use intervals of varying length $(h_k)_k$.

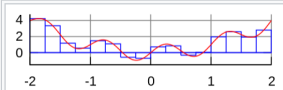


Illustration of the rectangle rule.

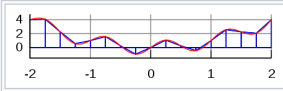


Illustration of the trapezoidal rule.

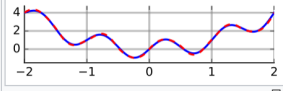


Illustration of Simpson's rule.