1204201

วิธีการเชิงตัวเลขสำหรับวิทยาการคอมพิวเตอร์
Numerical Method for Computer Science

ผู้ช่วยศาสตราจารย์ ดร.รพีพร ช่ำชอง

ภาควิชาวิทยาการคอมพิวเตอร์

คณะวิทยาการสารสนเทศ

มหาวิทยาลัยมหาสารคาม

1 Introduction

1.1 เนื้อหารายวิชา

1.1.1 คำอธิบายรายวิชา (Course Description)

ระบบสมการเชิงเส้นและเมตริกซ์ การประมาณค่าในช่วงและนอกช่วง การถดถอย แบบกำลังสองน้อยที่สุด การแก้สมการเชิงอนุพันธ์สามัญ การประมาณเส้นโค้ง การหาค่าที่เหมาะสม และการประยุกต์วิธีการเชิงตัวเลขในงานด้านวิทยาการคอมพิวเตอร์

Linear algebraic equation and matrix, interpolation and extrapolation, least square regression, ordinary differential equation, curve fitting, optimization and application of numerical method for computer science

จำนวนหน่วยกิต 3(2-2-5)

1.1.2 แผนการสอน

Week	Topic	Total (hr.)		Activities
		Theory	Lab	
1	Introduction to Numerical Method	2	2	Lecture Lab
2	Controlling error	2	2	Lecture Lab Practice
3-4	Linear system	4	4	Lecture Lab Practice
5-6	Curve fitting & Linear regression	4	4	Lecture Lab Practice
7	Nonlinear regression	2	2	Lecture Lab Practice

Week	Topic	Total (hr.)		Activities
		Theory	Lab	
8	Midterm Exam	3		
9-10	Interpolation	2	2	Lecture Lab Practice
11-12	Optimization	2	2	Lecture Lab
13-14	Apply numerical method for CS	4	4	Assignment Report Demo
15	Lab Exam		3	
16	Final Exam	3		

1.1.3 การประเมิน และเกณฑ์การประเมิน

วิธีการประเมิน	สัปดาห์ที่ประเมิน	สัดส่วนของการประเมิน
ฝึกแบบฝึกหัดและปฏิบัติในชั้นเรียน	1-7,9-12	30%
รายงาน/ นำเสนองาน/ demo ผลงาน	13-14	10%
สอบกลางภาค	8	25%
สอบปฏิบัติ	15	10
สอบกลางปลายภาค	16	25%

ประเมินผลตามเกณฑ์ที่กำหนด หรือ แบบ Normalized T Score

A	80 คะแนนขึ้นไป
B+	75-79.99
B	70-74.99
C+	65-69.99
C	60-64.99
D+	55-59.99
D	50-54.99
F	0-49.99

1.1.4 เอกสารและตำราหลัก

- Steven C. Chapra., Applied Numerical Methods with MATLAB® for Engineers and Scientists, 3rd ed., McGraw Hill, USA, 2012.
- Richard L. Burden and J. Douglas Faires, Numerical Analysis, 9th ed., Brooks/Cole, Cengage Learning, Boston, USA. 2011.
- Jaan Kiusalaas, Numerical methods in engineering with Python 3, Cambridge University Press, 2013.
- Jake VanderPlas, Python Data Science Handbook, O'Reilly Media, Inc., USA. 2017.
- เชี่ยวชาญการเขียนโปรแกรมด้วยไพธอน. ผู้ช่วยศาสตราจารย์ สุชาติ คุ้มมะณี
- ระเบียบวิธีเชิงตัวเลขในงานวิศวกรรม. ปราโมทย์ เดชะอำไพ
- ระเบียบวิธีเชิงตัวเลข. ธนาวุฒิ ประกอบผล
- วรสิทธิ์ กาญจนกิจเกษม. (2557). ระเบียบวิธีเชิงตัวเลข. สำนักพิมพ์แห่งจุฬาลงกรณ์มหาวิทยาลัย

1.2 Introduction to Numerical Method

วิธีการเชิงตัวเลข (Numerical Method) คือ การแก้ปัญหาฟังก์ชันทางคณิตศาสตร์ที่ต้องใช้คอมพิวเตอร์ช่วยคำนวณ

สำหรับเหตุผลที่นักคอมพิวเตอร์จำเป็นต้องเรียนเรื่องวิธีการเชิงตัวเลข เนื่องจาก

- ปัญหาคณิตศาสตร์บางอย่างไม่สามารถหาคำตอบได้โดยตรงจากการวิเคราะห์ เป็นการคำนวณหาคำตอบจากฟังก์ชัน ด้วยตัวเลขชุดใหม่ที่แทนลงในฟังก์ชัน ทำให้การคำนวณได้สะดวก และรวดเร็ว
- ปัญหาบางอย่างสามารถแก้ได้โดยระบบสมการเชิงเส้น สมการไม่เป็นเชิงเส้น เรขาคณิต โดยการใช้ แคลคูลัสเพื่อแก้ปัญหาได้
- วิธีการเชิงตัวเลขเป็นเครื่องมีที่มีประสิทธิภาพในการเรียนรู้เพื่อใช้คอมพิวเตอร์ในการออกแบบและพัฒนา แต่ขณะเดียวกันคอมพิวเตอร์มีข้อจำกัดที่ส่งผลต่อการคำนวณ ทำให้เกิดข้อผิดพลาด ซึ่งจำเป็นต้องหาวิธีควบคุมความผิดพลาดดังกล่าว
- เพื่อนำไปประยุกต์ใช้ในการประมาณค่าข้อมูลได้
- เพื่อเลือกวิธีการเชิงตัวเลขให้เหมาะสมกับปัญหามากที่สุด

สำหรับวิธีการแก้ปัญหาทางตัวเลขนั้น พบว่า

- ไม่มีวิธีการเชิงตัวเลขใดวิธีหนึ่งที่สามารถแก้ปัญหาได้ทุกชนิด
- ไม่มีวิธีการเชิงตัวเลขใดดีที่สุดสำหรับปัญหาทุกรูปแบบ
- ไม่มีวิธีการเชิงตัวเลขวิธีใดที่จะไม่ทำให้เกิดความคลาดเคลื่อนของผลลัพธ์ของการคำนวณ

การนำความรู้ด้านวิธีการเชิงตัวเลข ในปัจจุบันมีความจำเป็นอย่างมากและสามารถนำไปประยุกต์โดยใช้พื้นฐานดังกล่าวในงานด้านต่างๆ ดังต่อไปนี้

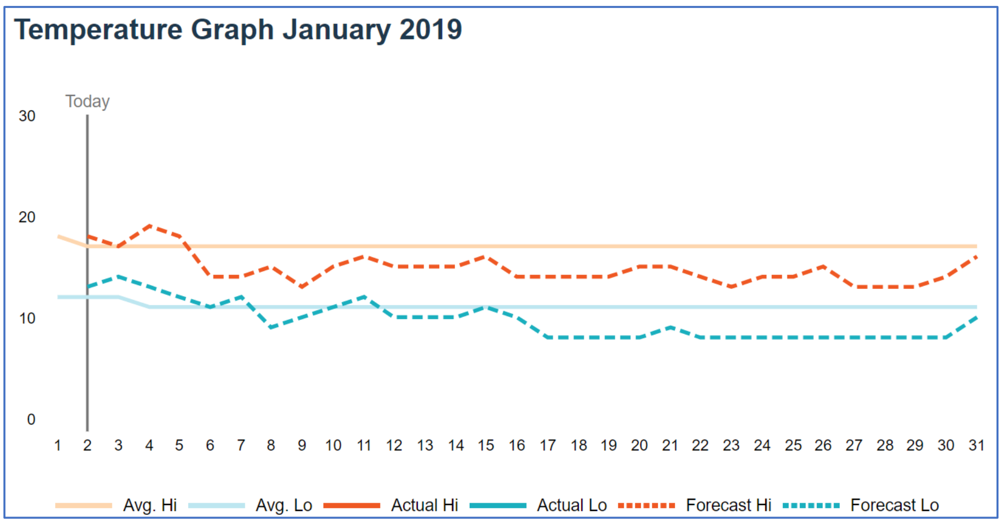
- งานด้านเทคโนโลยี/คอมพิวเตอร์ ได้แก่
- งานด้านวิทยาศาสตร์ข้อมูล (Data Science)
- งานด้านปัญญาประดิษฐ์ (Artificial Intelligence, AI)
- งานด้านอินเทอร์เน็ตทุกสรรพสิ่ง (Internet of Things, IoT)
- งานด้านข้อมูลขนาดใหญ่ (Big Data)
- งานด้านคอมพิวเตอร์กราฟฟิก (Computer Graphics)
- งานด้านการประมวลผลภาพและคอมพิวเตอร์วิทัศน์ (Image Processing and Computer Vision)
- การจำลองงานด้วยระบบคอมพิวเตอร์ (Simulation) ต่างๆ
- งานด้านการพยากรณ์ (Forecasting)
- งานด้านวิศวกรรม และวิทยาศาสตร์ เช่น
- การจำลองการยุบตัวของโครงสร้างรถยนต์ขณะเกิดการชน
- การจำลองการออกแบบลำตัวเครื่องบิน

- การจำลองงานด้านเคมี
- งานด้านสถิติ การเงิน เศรษฐศาสตร์ และการบัญชี เช่น
- การพยากรณ์ต่างๆ เช่น พยากรณ์อากาศ พยากรณ์น้ำไหลเข้าเขื่อน เป็นต้น
- การพยากรณ์ภาวะเศรษฐกิจของประเทศ
- การคำนวณเงินกู้ ดอกเบี้ย และระยะเวลาการผ่อนชำระ

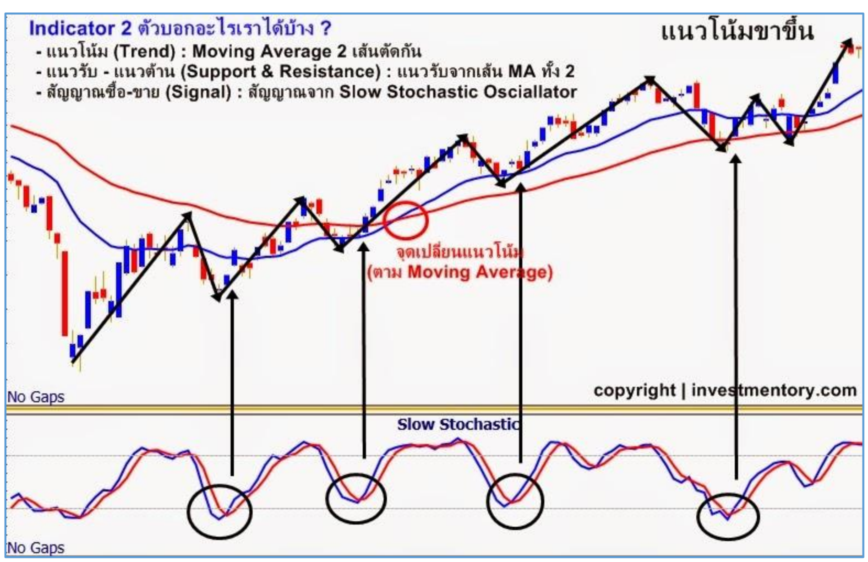
1.3 ตัวอย่างงานทางด้านคอมพิวเตอร์ที่เกี่ยวข้องกับ Numerical Method

1) การพยากรณ์อากาศ เช่น

https://www.accuweather.com/en/lb/beirut/227342/month/227342?view=table



2) การวิเคราะห์หุ้น



3) การลบรอยขีดเขียนบนรูปภาพ

4) การ Zoom-in หรือ Zoom-out รูปภาพ

5) การรู้จำลายนิ้วมือบนสมาร์ทโฟน หรือการรู้จำใบหน้า

https://www.gizbot.com/how-to/tips-tricks/learn-how-to-take-photos-using-fingerprint-scanner-on-any-android-smartphone-044506.html

6) โปรแกรม 3D Face Hologram Simulator

http://m.th.mobomarket.net/free-download-3d-face-hologram-simulator-4295310443.html

1.4 Introduction to Python3 for Numerical Method

สำหรับการเขียนโปรแกรมจัดการการแก้ปัญหาวิธีการเชิงตัวเลขในที่นี้ใช้โปรแกรมภาษา Python เนื่องจาก (Kiusalaas, 2013)

- Python เป็น Open-source ที่นักพัฒนาสามารถติดตั้งได้โดยไม่มีปัญหาด้านลิขสิทธิ์
- Python สามารถใช้งานได้ทั้งบนระบบปฏิบัติการ Linux Unix และ Windows iOS เป็นต้น โดยสามารถพัฒนาบนระบบปฏิบัติการใดปฏิบัติการหนึ่งโดยไม่ต้องแก้ไขโปรแกรม
- Python เป็นภาษาที่งานต่อการเรียนรู้มากกว่าภาษาอื่นๆ
- Python เป็นภาษาที่ง่ายต่อการติดตั้ง
- เป็นภาษาที่มีโครงสร้างพื้นฐานที่นำเอาจุดเด่นของภาษา JAVA ภาษา C++ รวมทั้งมีความคล้ายคลึงกับโปรแกรม MATLAB ที่นำมาประยุกต์ใช้ในการแก้ปัญหาทางคณิตศาสตร์ สามารถเขียนโปรแกรมในลักษณะของการเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming, OOP)

โปรแกรมภาษา Python ไม่สามารถคอมไพล์ (Compile) เป็นภาษาเครื่อง (Machine code) ได้ แต่จะทำงานโดยใช้ตัวแปลภาษาอินเทอร์พรีเตอร์ (Interpreter) ซึ่งสามารถแปลคำสั่งเป็น byte code ได้ ซึ่งจะช่วยให้การทำงานเร็วขึ้นโดยไม่ต้องคอมไพล์ นอกจากนี้ข้อดีของตัวแปลอินเทอร์พรีเตอร์ คือช่วยให้ทดสอบและดีบัก (Debug) โปรแกรมได้รวดเร็วขึ้น สามารถเลือกการทำงานเพียงคำสั่งที่ต้องการแล้วดูผล หรือรวมคำสั่งเฉพาะส่วน ซึ่งเมื่อเทียบกับภาษา Fortran และ C++ แล้วจะช่วยให้พัฒนาโปรแกรมได้รวดเร็วกว่า แต่อย่างไรก็ตามข้อเสียของอินเทอร์พรีเตอร์ คือ ไม่สามารถทำให้เป็น stand-alone application ได้ นั่นหมายความว่าโปรแกรม Python จะทำงานได้ก็ต่อเมื่อเครื่องดังกล่าวมีการติดตั้ง Python Interpreter นั่นเอง

Python เป็นภาษาที่มีการทำงานโดยไม่มีจุดจบของคำสั่ง แต่จะอาศัยการทำงานโดยโครงสร้างของ block ในการเขียนโปรแกรม เช่น block ภายใต้คำสั่งเงื่อนไข (Condition) การวนลูป (Loop) หรือฟังก์ชัน (Function)

1.4.1 โปรแกรมภาษา Python และ Package

สำหรับโปรแกรมภาษา Python ในเอกสารฉบับนี้ได้ Python version 3 ขึ้นไป โดยสามารถดาวน์โหลด (Download) Python Interpreter ได้จาก

http://www.python.org

สำหรับคำสั่งพื้นฐานที่ใช้ใน Python นั้นสามารถศึกษาได้จากเว็บไซต์ของ Python ได้โดยตรงซึ่งมีเอกสารการใช้งานคำสั่งต่างๆ มากมาย และแนะนำให้ศึกษาคำสั่งพื้นฐานได้จากหนังสือ

- เชี่ยวชาญการเขียนโปรแกรมด้วยไพธอน ของผู้ช่วยศาสตราจารย์สุชาติ คุ้มมะณี
- Python by Chris Fehly (Peachpit Press, CA, 2nd ed.)
- Python Essential Reference by David M. Beazley (Addison-Wesley, 4th ed.)
- A Primer on Scientific Programming with Python by Hans P. Langtangen (Springer-Verlag, 2009).
- http://www.python.org/doc/
- http://docs.python.org/tutorial/

สำหรับ Package ต่างๆ ที่ต้องใช้งานเมื่อติดตั้ง Python แล้ว มีการนำมาประยุกต์ใช้ในวิธีการเชิงตัวเลขนี้ ได้แก่

Package	แหล่างดาวน์โหลด	แหล่งเรียนรู้
scipy	http://www.scipy.org	
numpy	http://www.numpy.org/	http://www.scipy.org/Numpy_Example_List
matplotlib	https://matplotlib.org/	http://matplotlib.sourceforce.net/contents.html
pandas	http://pandas.pydata.org/	

1.4.2 Function และ Module

สำหรับฟังก์ชันใน Python มีโครงสร้างการทำงานดังนี้

```
def function_name(argu1, argu2, ...):
    statements
return return_values1, return_value2, ...
```

function_name คือ ชื่อฟังก์ชัน ตามรูปแบบการกำหนดชื่อซึ่งเป็น case sensitive

argu1, argu2, ... คือ การส่งผ่านค่าตัวแปรให้กับฟังก์ชันอาจจะมีหรือไม่ก็ได้

return_value1, return_value2, ... คือ การส่งค่ากลับออกไปนอกฟังก์ชัน จะมีหรือไม่ก็ได้ และสามารถส่งกลับได้มากกว่า 1 ค่า

statements คือ ชุดคำสั่ง

สำหรับการเรียกใช้ฟังก์ชัน ขึ้นกับการส่งผ่านค่าและการส่งค่ากลับ ดังนี้

กรณีที่ไม่มีการส่งผ่านค่า และไม่มีการส่งค่ากลับ

```
function_name( )
```

กรณีที่มีการส่งผ่านค่าแต่ไม่มีการส่งค่ากลับ

```
function_name(argu1, argu2, ...)
```

กรณีที่มีการไม่มีการส่งผ่านค่าแต่มีการส่งค่ากลับ

```
var1, var2, ... = function_name( )
```

กรณีที่มีการมีการส่งผ่านค่าและมีการส่งค่ากลับ

```
var1, var2, ... = function_name(argu1, argu2, ...)
```

ตัวอย่างการสร้างฟังก์ชันแบบต่างๆ

```
def display( ):
    print(‘Hello’)*3
display( )
```

```
def display(str):
    print(str)
display(‘Hello guy’)
```

```
def swap(a, b):
    tmp=a
    a=b
    b=tmp
    return a, b
a, b = swap(5,7)
```

1.4.3 Math Module

Math module เป็นการใช้งานคำสั่งในการคำนวณหลักพื้นฐานทางคณิตศาสตร์ การใช้งานคำสั่งจาก Math Module จะต้องเรียกใช้ module ดังนี้

```
from math import *
```

ศึกษาคำสั่งต่างๆ ได้จาก https://docs.python.org/3/library/math.html

สำหรับคำสั่งที่สำคัญที่จะกล่าวถึงในที่นี้ ได้แก่

ฟังก์ชัน	ความหมาย
math.ceil(x)	หาค่าฟังก์ชันเพดา (ค่าจำนวนเต็มที่น้อยที่สุดที่มากกว่าหรือเท่ากับค่า x)
math.floor(x)	หาค่าฟังก์ชันพื้น (ค่าจำนวนเต็มที่มากที่สุดที่น้อยกว่าหรือเท่ากับค่า x)
math.fabs(x)	หาค่าขนาด (absolute) ของ x
math.factorial(x)	หาค่าแฟคทอเรียล (factorial) ของ x
math.fmod(x,y)	หาเศษจากการหาร โดย x/y เหมือนกับคำสั่ง x%y แต่ฟังก์ชันนี้สามารถหาเศษของจำนวนจริง (floating-point) ได้ แต่ผลลัพธ์บางค่าอาจแตกต่างจากการใช้ %
math.fsum(iterable)	หาผลบวกของค่าที่อยู่ในรูปลำดับ
math.gcd(a,b)	หาค่าหารร่วมมาก (Greatest Common Division) ระหว่าง a กับ b
math.exp(x)	หาค่า exponential (e^x) ซึ่ง e = 2.718281
math.log(x[, base])	หาค่า log ซึ่ง ถ้าส่งผ่านค่า x ตัวเดียวจะเป็นการหา natural logarithm (ฐาน e) ถ้าส่งผ่านค่า x และ base เป็นการหาค่า log ฐาน base
math.log10(x)	หาค่า log x ฐาน 10
math.pow(x,y)	หาค่า x^y
math.sqrt(x)	หาค่ารากที่ 2 ของ x ($\sqrt{x}$)
math.cos(x)	หาค่า cosine ของเรเดียน x

ฟังก์ชัน	ความหมาย
math.sin(x)	หาค่า sine ของเรเดียน x
math.tan(x)	หาค่า tangent ของเรเดียน x
math.acos(x)	หาค่า arc cosine ของเรเดียน x
math.asin(x)	หาค่า arc sine ของเรเดียน x
math.atan(x)	หาค่า arc tangent ของเรเดียน x
math.degrees(x)	แปลงมุม x จากเรเดียนเป็นองศา
math.radians(x)	แปลงมุม x จากองศาเป็นเรเดียน

ค่าคงที่

ค่าคงที่	ความหมาย
math.pi	ค่า $\pi = 3.141592 \ldots$
math.inf	ค่าจำนวนจริงบวกอนันต์ (infinity) เหมือนกับ float('inf')
math.e	ค่า exponential (e) = 2.718281...
math.nan	ไม่ใช่ตัวเลข (NaN, "Not a number") เหมือนกับ float('nan')

1.4.4 NumPy Module

เนื่องจากในภาษา Python เป็นการจัดการข้อมูลกับโครงสร้างของ list, dictionary, tuple เป็นหลัก ซึ่งไม่สามารถคำนวณได้โดยอัตโนมัติ ดังนั้นการใช้ numpy ซึ่งเป็น module ที่มีวัตถุประสงค์เพื่อ

1. การคำนวณกับข้อมูลที่เป็นอาร์เรย์
2. การคำนวณที่มีประสิทธิภาพกับข้อมูลอาร์เรย์หลากหลายมิติ
3. ออกแบบสำหรับการคำนวณทางวิทยาศาสตร์ และวิศวกรรมศาสตร์

ตัวอย่างการใช้งานคำสั่งเกี่ยวกับ NumPy module ศึกษาเพิ่มเติมได้ที่

https://www.python-course.eu/numpy.php

http://www.scipy.org/Numpy_Example_List

ในการเรียกใช้คำสั่ง NumPy จะต้อง import โดยใช้คำสั่ง

```
import numpy
```

หรือนิยมอ้างอิง numpy ด้วย np

import numpy as np

ในที่นี้จะใช้การอ้างอิงแบบที่ 2 ด้วยการอ้างอิง np สำหรับคำสั่งต่างๆ ใน numpy ที่จะใช้ในวิชานี้

ลักษณะเด่นที่สำคัญของ Numpy สามารถแสดงให้เห็นได้ดังตัวอย่างต่อไปนี้

```
#Lab1_01.ipynb

import numpy as np

temperature=[20.5, 20.4, 21.3, 22.2, 22.8, 22.6, 21.2, 21.3, 20.4, 20.9]
print('temperature=',temperature)
fahrenheit=temperature * 9 / 5 + 32
print('Fahrenheit=',fahrenheit)
```

```
temperature= [20.5, 20.4, 21.3, 22.2, 22.8, 22.6, 21.2, 21.3, 20.4, 20.9]

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-8c350f999b12> in <module>()
      5 temperature=[20.5, 20.4, 21.3, 22.2, 22.8, 22.6, 21.2, 21.3, 20.4, 20.9]
      6 print('temperature=',temperature)
----> 7 fahrenheit=temperature * 9 / 5 + 32
      8 print('Fahrenheit=',fahrenheit)

TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

จากตัวอย่างข้างต้นจะเห็นว่าหากนำ list มาคำนวณโดยตรงจะไม่สามารถทำได้ หากแก้ไขโปรแกรมใหม่ โดยการแปลงตัวแปร list ให้อยู่ในรูป array ของ module Numpy โดยใช้คำสั่ง np.array ดังตัวอย่าง ดังต่อไปนี้

```
celsius = np.array(temperature)
print('Celsius=',celsius)

fahrenheit=celsius * 9 / 5 + 32
print('Fahrenheit=',fahrenheit)
```

```
Celsius= [20.5 20.4 21.3 22.2 22.8 22.6 21.2 21.3 20.4 20.9]
Fahrenheit= [68.9  68.72 70.34 71.96 73.04 72.68 70.16 70.34 68.72 69.62]
```

รูปแบบของ format หากมี [ ] หมายถึงไม่บังคับให้ใส่ และหากมีการกำหนดค่าเริ่มต้นจะมีเครื่องหมาย = กำหนดค่า default ให้ ดังนั้นจึงไม่จำเป็นต้องกำหนดค่า

คำสั่งต่างๆ ใน NumPy ที่มีใช้ในวิชานี้ได้แก่

1.4.4.1 ค่าแอททริบิว (Attribute)

- dtype เป็นการแสดงชนิดของข้อมูลของสมาชิกใน array เช่น

```
>>> x
array([[0, 1],
       [2, 3]])
>>> x.dtype
dtype('int32')
```

- itemsize เป็นการแสดงขนาดหน่วยความจำของสมาชิกแต่ละตัวใน array หน่วยเป็น byte เช่น

```
>>> x = np.array([1,2,3], dtype=np.float64)
>>> x.itemsize
8
>>> x = np.array([1,2,3], dtype=np.complex128)
>>> x.itemsize
16
```

- ndim เป็นการหาจำนวนมิติ (dimension) ของ array

```
>>> x=np.array([1,2,3])
>>> x.ndim
1
>>> x=np.array([[1, 2], [2, 3], [3, 4]])
>>> x.ndim
2
```

- shape เป็นการหาขนาดของ array ในแต่ละมิติ

```
>>> x=np.array([1, 2, 3])
>>> x.shape
(3,)
>>> x=np.array([[1, 2], [2, 3], [3, 4]])
>>> x.shape
(3, 2)
```

- size เป็นการหาจำนวนสมาชิกของ array

```
>>> x=np.array([1, 2, 3])
>>> x.size
3
>>> x=np.array([[1, 2], [2, 3], [3, 4]])
>>> x.size
6
```

- T เป็นการ Transpose เมทริกซ์ โดยแปลงแถวเป็นหลัก และหลักเป็นแถว

```
>>> x = np.array([[1., 2.],[3., 4.]])
>>> x
array([[ 1.,  2.],
       [ 3.,  4.]])
>>> x.T
array([[ 1.,  3.],
       [ 2.,  4.]])
```

- นอกจากนี้ยังมี attribute อื่นๆ อีก ได้แก่ flags, flat, strides, real และ imag

1.4.4.2 การกำหนดชนิดของข้อมูล (data-type)

ในการกำหนดชนิดข้อมูลของ NumPy มีดังต่อไปนี้

ชนิดข้อมูล	ฟังก์ชันชนิดข้อมูล	ชนิดข้อมูล และจำนวนบิต (bits)
int8	numpy.int8( )	integer มีเครื่องหมาย ขนาด 8 บิต
int16	numpy.int16( )	integer มีเครื่องหมาย ขนาด 16 บิต
int32	numpy.int32( )	integer มีเครื่องหมาย ขนาด 32 บิต
int64	numpy.int64( )	integer ไม่มีเครื่องหมาย ขนาด 64 บิต
uint8	numpy.uint8( )	integer ไม่มีเครื่องหมาย ขนาด 8 บิต
uint16	numpy.uint16( )	integer ไม่มีเครื่องหมาย ขนาด 16 บิต
uint32	numpy.uint32( )	integer ไม่มีเครื่องหมาย ขนาด 32 บิต
uint64	numpy.uint64( )	integer ไม่มีเครื่องหมาย ขนาด 64 บิต

ชนิดข้อมูล	ฟังก์ชันชนิดข้อมูล	ชนิดข้อมูล และจำนวนบิต (bits)
float32	numpy.float32( )	จำนวนจริง ขนาด 32 บิต
float64	numpy.float64( )	จำนวนจริง ขนาด 64 บิต
bool_	numpy.bool_( )	บูลีน

1.4.4.3 การสร้าง array จากโครงสร้างข้อมูลที่มีอยู่แล้ว

- numpy.array : เป็นคำสั่งแปลง list และ tuple ให้เป็น array

```
array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)
```

พารามิเตอร์ (Parameters) ในที่นี้จะใช้เฉพาะพารามิเตอร์ 2 ตัวแรก และตัวสุดท้ายเท่านั้น:

object : โครงสร้างข้อมูลที่เป็นแบบลำดับ เช่น list หรือ tuple เป็นต้น

dtype : data-type (optional) หากไม่ระบุชนิดข้อมูลโปรแกรมจะทำนายค่าชนิดข้อมูลให้อัตโนมัติ

ndmin : int (optional) เป็นการกำหนดจำนวนมิติ (dimension) ของ array

ค่าส่งกลับ (Return): out : ndarray

```
>>> np.array([1, 2, 3])
array([1, 2, 3])
```

ตัวอย่างการกำหนดค่าเป็นทั้ง int และ float ค่าที่ return จะส่งกลับมาเหมือนค่าที่มีขนาดใหญ่กว่า คือ casting int ให้เป็น float โดยอัตโนมัติ

```
>>> np.array([1, 2, 3.0])
array([ 1.,  2.,  3.])
```

ตัวอย่างการกำหนดค่ามากกว่า 1 มิติ

```
>>> np.array([[1, 2], [2, 3]])
array([[1, 2],
       [3, 4]])
```

ตัวอย่างการกำหนด จำนวน dimension เป็น 2 มิติ

```
>>> np.array([1, 2, 3], ndmin=2)
array([[1, 2, 3]])
```

1.4.4.4 การสร้าง Array ใหม่

- numpy.arange : เป็นการสร้าง array จากการกำหนดช่วงของข้อมูลโดยระบุจุดเริ่มต้นและจุดสิ้นสุด

arange([start,] stop[, step], [, dtype=None])

Parameters:

start : number (optional) เป็นตัวเลขเริ่มต้น default = 0 ถ้าไม่กำหนด

stop : number เป็นค่าสิ้นสุด โดยไม่รวมค่านี้ยกเว้นกรณีที่ค่า step ไม่ใช่ integer และ floating point ที่มีการทำ round-off

step : number (optional) เป็นระยะห่างระหว่างตัวเลขแต่ละตัว default = 1 ถ้าไม่กำหนด

dtype : การกำหนดชนิดของข้อมูล ค่า default = None หมายถึงค่าเหมือนกับ input argument

ค่าส่งกลับ (Return): out : ndarray

```
>>> np.arange(3)
array([0, 1, 2])
>>> np.arange(3.0)
array([ 0.,  1.,  2.])
>>> np.arange(3, 7)
array([3, 4, 5, 6])
>>> np.arange(3, 7, 2)
array([3, 5])
```

- numpy.linspace: เป็นการกำหนดค่าในช่วงตามจำนวนตัวเลขที่กำหนด โดยฟังก์ชันนี้จะแบ่งข้อมูลตามจำนวนในช่วงให้โดยอัตโนมัติ ที่มีระยะห่างเท่ากัน

linspace(start, stop, num=50, endpoint=True, retstep=False , dtype=None)

Parameters:

start : ค่าเริ่มต้น.

stop : ค่าสิ้นสุด ซึ่งจะรวมอยู่ด้วยหาก endpoint ไม่เป็น False

num : int ต้องไม่เป็นลบ (optional) ค่า default = 50

endpoint : bool (optional) default=True ซึ่ง stop คือค่าตัวเลขตัวสุดท้าย แต่ถ้าเป็น False จะไม่รวมค่า stop retstep : bool (optional) default=False ถ้าหากเป็น True จะส่งค่ากลับ 2 ค่าคือ (samples, step) ซึ่ง step คือระยะห่างระหว่างค่าแต่ละค่า

dtype : dtype (optional)

Returns:

samples : ndarray

ถ้าหาก restep = True จะ return ดังนี้

step : float

- numpy.zeros: คำสั่งสร้าง array โดยกำหนดให้ค่าเริ่มต้นใน array เป็น 0
- numpy.ones: คำสั่งสร้าง array โดยกำหนดให้ค่าเริ่มต้นใน array เป็น 1

```
zeros(shape, dtype = None, order = 'C')
ones(shape, dtype = None, order = 'C')
```

Parameters:

shape: ขนาดของ array

dtype: ชนิดของข้อมูล

Return: array ที่มีค่าในสมาชิกทุกตัวเป็น 0 สำหรับ zeros และ 1 สำหรับ 0

- numpy.full: คำสั่งสร้าง array โดยกำหนดค่าเริ่มต้น array ให้ตามที่กำหนด

```
full(shape, fill_value, dtype=None, order='C')
```

Parameters:

shape: ขนาดของ array

dtype: ชนิดของข้อมูล

fill_value: ค่าคงที่ที่กำหนดให้สมาชิกของ array

Return: array ที่มีค่าในสมาชิกทุกตัวตามค่า fill_value

- numpy.eye: =

```
eye(N, M=None, k=0, dtype=<class 'float'>, order='C')
```

Parameters:

N : จำนวนบรรทัดของ array

M : จำนวนคอลัมน์ของ array (optional) ถ้าไม่กำหนด (default=None) นั่นคือ M=N k : จำนวนเต็ม (optional) มีค่า default=0 นั่นคือกำหนดค่า 1 ในแนวทะแยง (diagonal) หลัก แต่ถ้า k มีค่าเป็นบวก เป็นการกำหนดค่า 1 ในแนวทะแยงด้านบนเพิ่มทีละลำดับ หรือ k มีค่าเป็นลบ เป็นการกำหนดค่า 1 ในแนวทะแยงด้านล่างลดทีละลำดับ

dtype: ชนิดของข้อมูล

order : {'C', 'F'} (optional) C=C-style ลำดับแบบบรรทัดแล้วตามด้วยคอลัมภ์ ส่วน F = Fortran-style เก็บตามลำดับของหน่วยความจำ

Returns:

I : ndarray ขนาด (N,M)

- numpy.empty: สร้าง array โดยไม่กำหนดค่าเริ่มต้น

```
empty(shape, dtype=float, order='C')
```

- numpy.random.random: สร้าง array โดยสุ่มค่าตัวเลขให้

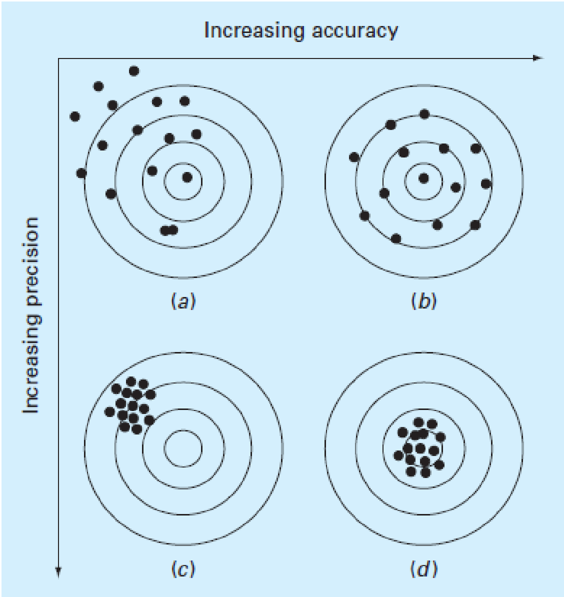
```
random.random(size=None)
```

2 การควบคุมความผิดพลาด (Controlling Error)

2.1 Why controlling errors?

1) Computer errors
2) เพื่อหาความแม่นยำ (accuracy) และความเที่ยงตรง (precision) ของผลลัพธ์เชิงตัวเลข
3) เพื่อพัฒนาหา criteria ในการหยุดอัลกอริทึมในการทำซ้ำ (iterative algorithms) เนื่องจากข้อจำกัดของระบบคอมพิวเตอร์

2.2 ความแม่นยำ (Accuracy) และความเที่ยงตรง (Precision)



ภาพที่ 2.1 ตัวอย่างการยิงปืนที่แสดงความแม่นยำและความเที่ยงตรง (a) ไม่แม่นยำและไม่เที่ยง (b) แม่นยำแต่ไม่เที่ยงตรง (c) ไม่แม่นยำแต่เที่ยงตรง (d) ทั้งแม่นยำและเที่ยงตรง

2.3 การแทนค่าข้อมูลตัวเลข (Number representation)

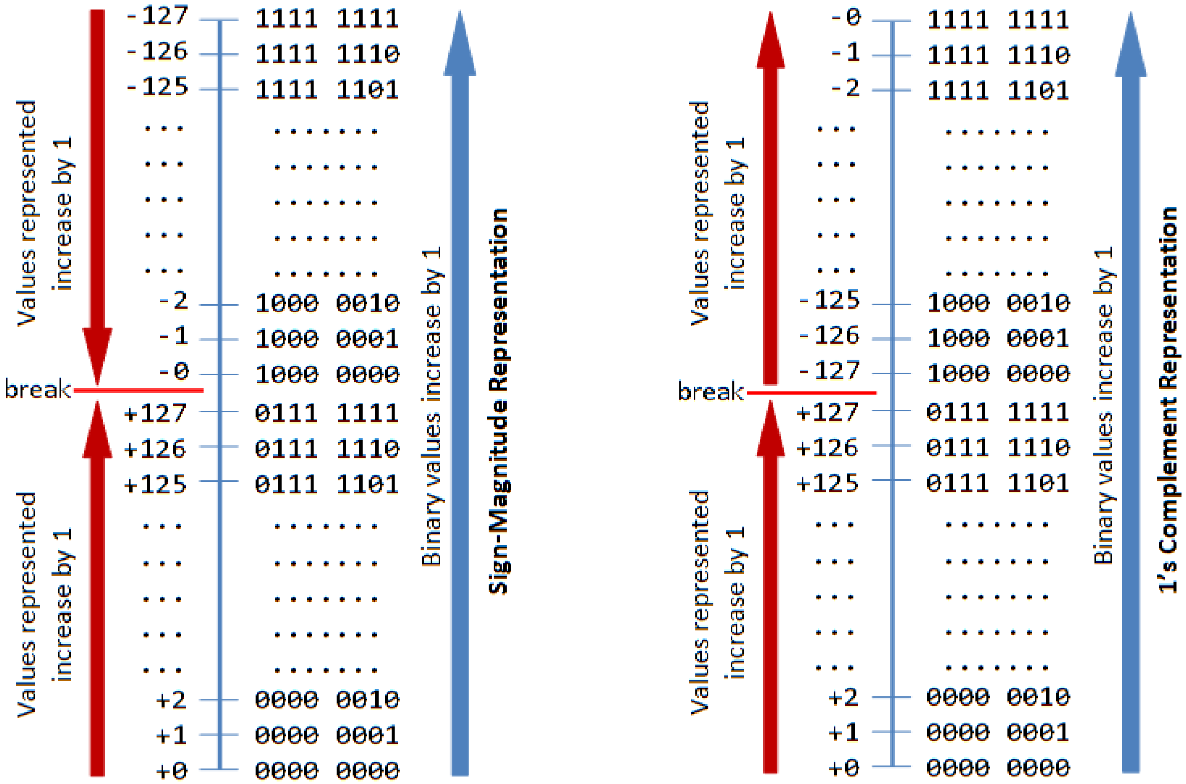
2.3.1 การแทนค่าข้อมูลจำนวนเต็ม (Integer representation)

- การแทนค่าจำนวนเต็มแบบไม่มีเครื่องหมาย (Unsigned representation)

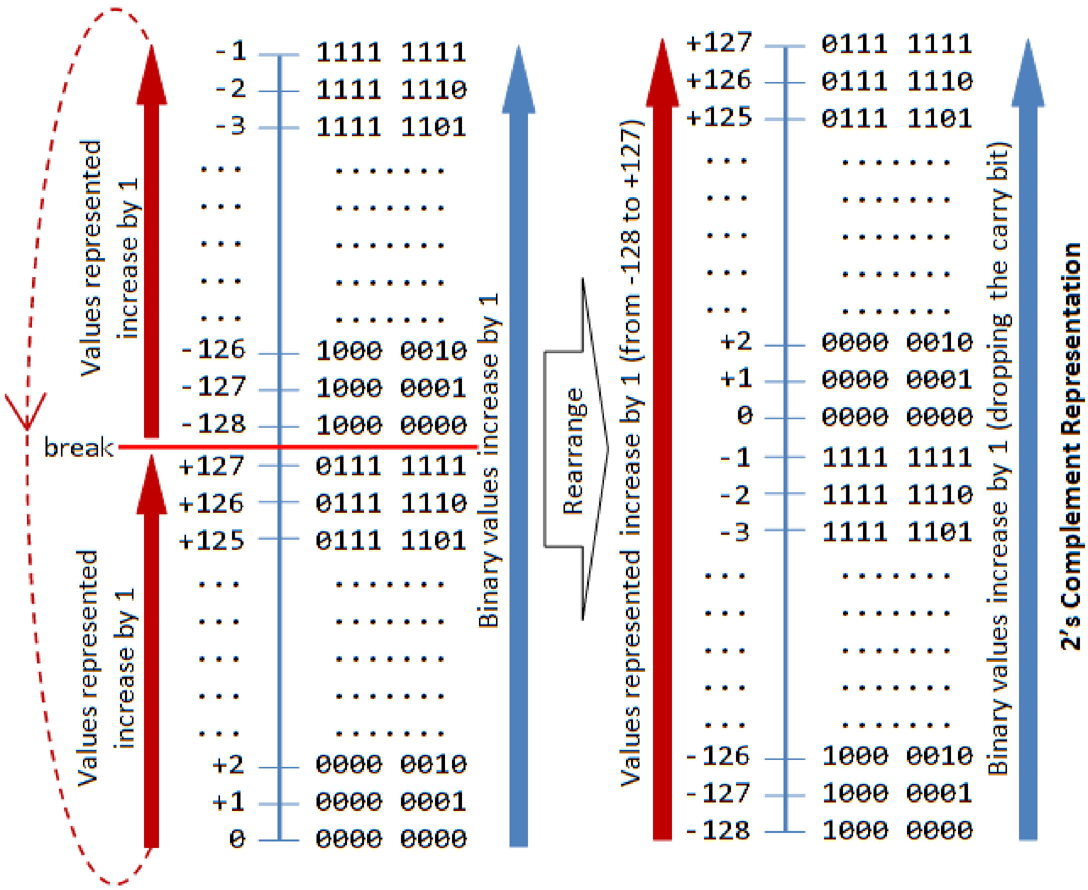
จำนวนบิต n บิต สามารถแทนข้อมูลได้ 2^n มีค่าอยู่ระหว่าง 0 ถึง $2^n - 1$

n	จำนวนข้อมูล	Min	Max
8	2^8	0	$2^8 - 1 = 255$
16	2^{16}	0	$2^{16} - 1$ =65,535
32	2^{32}	0	$2^{32} - 1$ =4,294,967,295 (9+ digits)
64	2^{64}	0	$2^{64} - 1$ =18,446,744,073,709,551,615 (19+ digits)

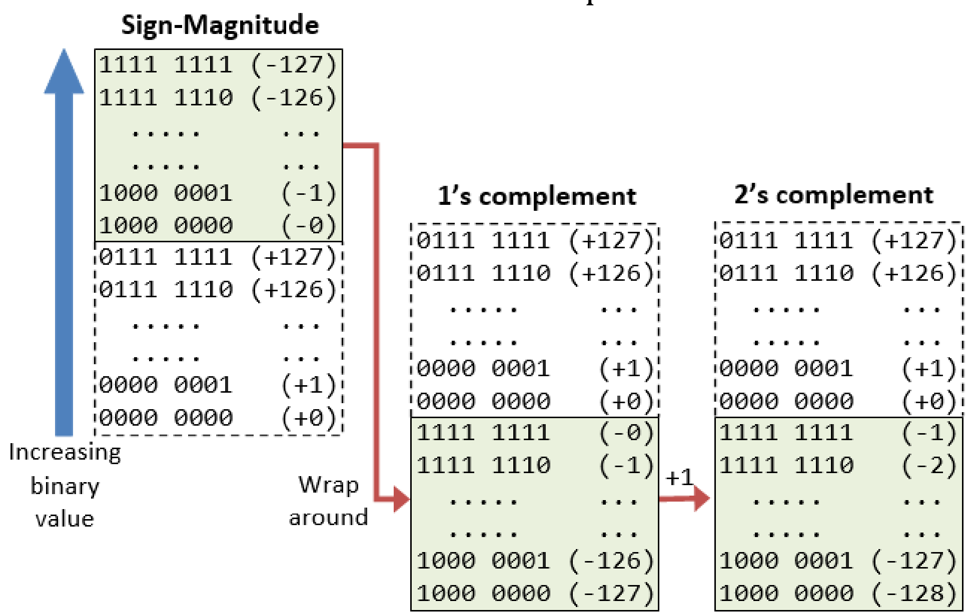
- การแทนค่าจำนวนเต็มแบบมีเครื่องหมาย (Signed representation)
- แบบเครื่องหมายขนาด (Signed magnitude)
- แบบเติมเต็มหนึ่ง (One's complement)
- แบบเติมเต็มสอง (Two's complement)



ภาพที่ 2.2 การแทนค่าแบบเครื่องหมายขนาด และแบบ 1's complement ขนาด 8 บิต
ที่มา:
https://www3.ntu.edu.sg/home/ehchua/programming/java/datarepresentation.html



ภาพที่ 2.3 การแทนค่าแบบ 2's complement ขนาด 8 บิต



ภาพที่ 2.4 เปรียบเทียบการแทนค่าจำนวนเต็มแบบมีเครื่องหมายขนาด 8 บิต

ในการแทนค่าข้อมูลแบบมีเครื่องหมายสำหรับระบบคอมพิวเตอร์จะใช้ 2's complement เนื่องการ

1) การแทนค่า 0
2) ปัญหาการบวกและลบเลขจำนวนเต็ม

ตัวอย่างที่ 2.1: บวกเลขจำนวนเต็มบวก 2 จำนวน กำหนดจำนวนบิต = 8
65 + 5 = 70 จะเห็นว่าผลบวกที่ได้ถูกต้อง

Decimal	**Binary**		
	2's complement	**1's complement**	**Signed magnitude**
65	0100 0001	0100 0001	0100 0001
5	0000 0101+	0000 0101+	0000 0101+
70	0100 0110 → 70	0100 0110 → 70	0100 0110 → 70

ตัวอย่างที่ 2.2: ลบเลขจำนวนเต็ม 2 จำนวน ซึ่งเป็นการบวกจำนวนเต็มบวกกับจำนวนเต็มลบ กำหนดจำนวนบิต = 8 โดย 5 - 5 = 65 + (-5) = 60 ซึ่งผลลัพธ์ที่ได้ถูกต้อง

Decimal	**Binary**		
	2's complement	**1's complement**	**Signed magnitude**
65	0100 0001	0100 0001	0100 0001
-5	1111 1011+	1111 1010+	1000 0101+
60	0011 1100 → 60	0011 1011 → ~~59~~	1100 0110 → ~~-70~~

ตัวอย่างที่ 2.3: บวกเลขจำนวนเต็มลบ 2 จำนวน กำหนดจำนวนบิต = 8
-65 - 5 = (-65) + (-5) = -70 ซึ่งผลลัพธ์ที่ได้ถูกต้อง

Decimal	**Binary**		
	2's complement	**1's complement**	**Signed magnitude**
-65	1011 1111	1011 1110	1100 0001
-5	1111 1011+	1111 1010+	1000 0101+
-70	1011 1010 → -70	1011 1000 → -71	1100 0110 → -70

อย่างไรก็ตามถึงแม้คอมพิวเตอร์จะใช้ 2's complement แต่ข้อจำกัดของขนาดของข้อมูลในการบวกและลบเลขจะมีโอกาสเกิด Overflow และ Underflow ได้ ดังตัวอย่างต่อไปนี้

ตัวอย่างที่ 2.4: กำหนดจำนวนบิต = 8 เช่น 127 + 2 = 129

จะเห็นว่าผลลัพธ์ที่ได้เกินขนาด ทำให้เกิด Overflow

Decimal	Binary-2's complement	Comment
127	0111 1111	
2	0000 0010+	
129	1000 0001 → ~~-127~~	ผิดเพราะผลบวกเกินช่วงจึงวนกลับ

ตัวอย่างที่ 2.5: กำหนดจำนวนบิต = 8 เช่น -125 - 5 = -130

จะเห็นว่าผลลัพธ์ที่ได้ต่ำกว่าขนาด ทำให้เกิด Underflow - below the range

Decimal	Binary-2's complement	Comment
125	1000 0011	
-5	1111 1011+	
-130	0111 1110 → ~~+126~~	ผิดเพราะผลบวกต่ำกว่าช่วงจึงวนกลับ

สำหรับช่วงข้อมูลของ 2's complement อยู่ระหว่าง -2^{n-1} ถึง $2^{n-1}-1$

n	Min	Max
8	$-2^7 = -128$	$2^7 - 1 = 127$
16	$-2^{15} = -32{,}768$	$2^{15} - 1 = 32{,}767$
32	$-2^{31} = -2{,}147{,}483{,}648$	$-2^{31} - 1 = 2{,}147{,}483{,}647$ (9+ digits)
64	$-2^{63} =$ $-9{,}223{,}372{,}036{,}854{,}775{,}808$	$-2^{63} - 1$ $= 9{,}223{,}372{,}036{,}854{,}775{,}807$

สำหรับโปรแกรมภาษาต่างๆ เช่น C/C++ และ Java การกำหนดชนิดข้อมูลจะถูกจำกัดขนาด เช่น int จะมีขนาด 32 บิต และ long int มีขนาด 64 บิต เป็นต้น แต่สำหรับภาษา Python3 ตัวเลขจำนวนเต็มจะมีขนาดไม่จำกัด สามารถกำหนดได้ตามขนาดของหน่วยความจำ ส่วน Python2 ยังมีข้อจำกัดอยู่ และนอกจากนี้หากมีการใช้การคำนวณจากโมดูลต่างๆ เช่น scipy numpy และ pandas ก็ยังมีการจัดเก็บแบบ C-style

2.3.2 การแทนค่าข้อมูลจำนวนจริงตามมาตรฐาน IEEE-754 (IEEE-754 floating-point number representation)

- S = Sign bit = {0, 1}
- B = Based exponent
- E = Exponent
- F = Fraction/ Mantissa

กรณีการแทนค่า 32 บิต Single-precision floating-point

MSB		LSB
31	30 23	22 0
Sign (S)	Exponent (E)	Fraction (F)
1 บิต	8 บิต	23 บิต

กรณีการแทนค่า 64 บิต Double-precision floating-point

MSB		LSB
63	62 52	51 0
Sign (S)	Exponent (E)	Fraction (F)
1 บิต	11 บิต	52 บิต

- Sign → จำนวนบวก : 0
  → จำนวนลบ : 1
- Exponent → Single precision (8 bits): Bias ของ $2^7 - 1 = 127$
  → Double precision (11 bits): Bias ของ $2^{10} - 1 = 1023$
- Fraction → ค่านัยสำคัญ Single precision (23 bits)
  Double precision (52 bits)
  ในรูปแบบ “Normalized Form”

$$R = \pm F \times B^{\pm E}$$

ตัวอย่างที่ 2.6: การแทนค่า Floating point ด้วย IEEE 754

1) แปลงเลขฐานสิบเป็นฐานสอง

$$10.625 \;= (1010.101)_2$$

2) จัด Normalize Form กับเลขฐานสอง โดยให้อยู่ในรูป

$$\pm(1.ddddd \ldots)_2 \times 2^{\pm E}$$

เช่น $(1010.101)_2 \times 2^0 \rightarrow (1.010101)_2 \times 2^3$

$-(0.0001101)_2 \times 2^0 \rightarrow -(1.101)_2 \times 2^{-4}$

3) นำค่า E (exponent) มาหาค่า Bias Exponent ในรูปฐานสอง เช่น

- Single precision: Bias = 127 เช่น

$(1.010101)_2 \times 2^3$ ซึ่ง $E = 3$ แล้ว

$$Bias\ Exponent = 127 + 3 = 130 = (1000\ 0010)_2$$

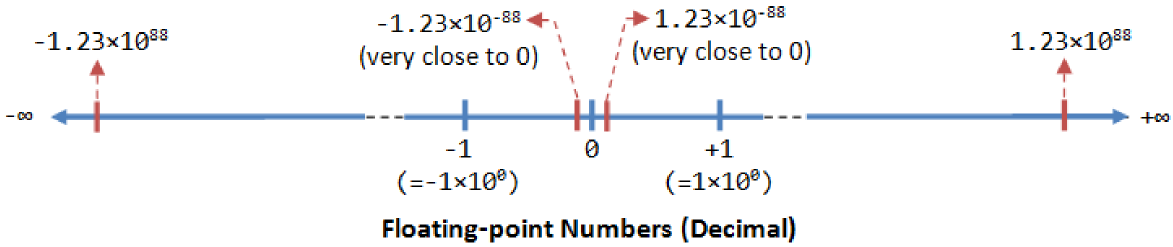
$-(1.101)_2 \times 2^{-4}$ ซึ่ง $\mathrm{E} = -4$ แล้ว

$$Bias\ Exponent = 127 + (-4) = 123 = (0111\ 1011)_2$$

Binary Value	Normalize As	Exponent	Biased Exponent (127 + Exponent)
-1.11	-1.11	0	127
1 01111111 11000000000000000000000			
+1101.101	+1.101101	+3	130
0 10000010 10110100000000000000000			
-.00101	-1.01	-3	124
1 01111100 01000000000000000000000			
+100111.0	+1.001110	+5	132
0 10000100 00111000000000000000000			
+.0000001101011	+1.101011	-7	120
0 01111000 10101100000000000000000			

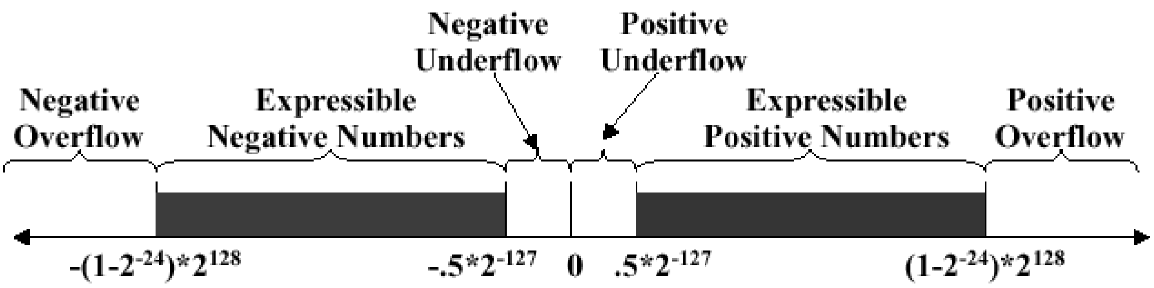
2.3.2.1 *การเกิด Overflow/Underflow สำหรับการแทนค่าเลข Floating point*

สำหรับรูปแบบทั่วไปของ Floating point ในระบบเลขฐานสิบ แสดงดังภาพ



ภาพที่ 2.5 รูปแบบระบบตัวเลขจำนวนจริงในระบบฐานสิบ

แต่เมื่อทำการแทนค่าข้อมูลตัวเลขจำนวนจริงด้วยระบบคอมพิวเตอร์ตามมาตรฐาน IEEE-754 แล้วก็อาจจะเกิด Overflow/Underflow ขึ้นได้ ดังภาพ



ภาพที่ 2.6 การเกิด Overflow/Underflow สำหรับการแทนค่าเลข Floating point

การแก้ปัญหาการเกิด Overflow/Underflow สำหรับการแทนค่าเลข Floating point ขึ้นกับระบบคอมพิวเตอร์ มีวิธีจัดการ 2 วิธี คือ

- การปัดเศษ (Rounding) จะดำเนินการกับ Fraction โดยปัดหลักที่ $n+1$ ขึ้นไปที่หลักที่ n
- การตัดทิ้ง (Chopping) จะดำเนินการกับ Fraction โดยตัดทิ้งหลักที่ $n+1$ โดยไม่สนใจว่าจะมีค่าเท่าไร

เลขทศนิยม	**วิธีการ**	
	ปัดเศษ	**ตัดทิ้ง**
0.33988	0.3399×10^0	0.3398×10^0
$\frac{2}{3}$	0.67×10^0	0.66×10^0
0.000415	0.42×10^{-3}	0.41×10^{-3}

สำหรับใน Python การจัดเก็บเลขทศนิยม จะเก็บในรูปของ scientific ดังตัวอย่าง

```
x=1.0**50
print(x)
y=10.0**200
print(y)
```

```
1.0
1e+200
```

2.4 ความคลาดเคลื่อน (Errors)

เกิดจากการวัดต่างๆ พบว่าเครื่องมือวัดมีขีดจำกัด ทำให้เกิดความคลาดเคลื่อน (errors) ขึ้นได้ มีปัจจัยจาก

- เครื่องมือวัด
- วิธีการวัด, ประสบการณ์ผู้วัด ถือเป็น human error
- ข้อมูลที่ใช้คำนวณ (data error)
- ฯลฯ

2.4.1 ประเภทของการเกิดความคลาดเคลื่อน ได้แก่

1) เกิดจากแบบจำลองทางคณิตศาสตร์ (Mathematical modeling)
2) เกิดจากความเผอเรอ (Mistake of blunder)
3) เกิดจากข้อมูล (Data error)
4) เกิดจากการตัดทศนิยมทิ้ง (Truncation error) หรือการปัดเศษขึ้น (Rounded-off error)
5) เกิดจากการแพร่กระจายความคลาดเคลื่อน (Propagation of error)

2.4.1.1 ความคลาดเคลื่อนจากแบบจำลองทางคณิตศาสตร์ (Mathematical modeling)

เนื่องจากแบบจำลองทางคณิตศาสตร์ อาศัยข้อมูลจริงที่เก็บรวบรวมได้ และจำลองตัวแบบทางคณิตศาสตร์โดยใช้คอมพิวเตอร์ ทำให้เกิดความผิดพลาดขึ้นได้ เพราะ

- ข้อมูลจริงที่รวบรวมได้มีความละเอียดมาก เพื่อให้ผลลัพธ์ที่มีความเที่ยงตรงมาก
- คอมพิวเตอร์ไม่สามารถเก็บข้อมูลลงหน่วยความจำได้ ทำให้เกิดความคลาดเคลื่อนยิ่งสูง

ตัวอย่างเช่น การจำลองสภาวะอากาศที่ไหลผ่านเครื่องบิน

2.4.1.2 ความคลาดเคลื่อนจากความเผอเรอ (Mistake of blunder)

- เกิดจากมนุษย์หรือผู้คำนวณเอง

1) ใส่ตัวเลขผิดพลาด
2) ใส่เครื่องหมายผิดพลาด
3) ใส่ตำแหน่งทศนิยมผิดพลาด
4) เขียนโปรแกรมผิดพลาด

2.4.1.3 ความคลาดเคลื่อนจากข้อมูล (Data error)

เกิดจากการเก็บรวบรวมข้อมูลที่อาจผิดพลาด ส่งผลให้การหาค่าที่จะนำมาใช้ผิดพลาด เช่น การคำนวณหาค่าเฉลี่ย ค่าส่วนเบี่ยงเบนมาตรฐาน ค่าร้อยละ เป็นต้น

2.4.1.4 ความคลาดเคลื่อนจากการตัดทศนิยมทิ้ง (Truncation error) หรือการปัดทศนิยมขึ้น (Rounded-off error)

- เกิดจากการคำนวณโดยมนุษย์และกำหนดตำแหน่งทศนิยมน้อยเกินไป
- เกิดจากข้อจำกัดของการแทนค่าข้อมูลในระบบคอมพิวเตอร์

  เช่น 0.557355546666 แต่นำมาใช้เพียง 3 ตำแหน่งคือ 0.557 เป็นต้น

2.4.1.5 ความคลาดเคลื่อนจากการแพร่กระจาย (Propagation of error)

เกิดจากการคูณหรือหารจำนวนที่มีค่ามากๆ หรือน้อยๆ เช่น

$$x_i = x_i^{10} * 0.55555555555$$

$$y_i = \frac{0.0000001111}{y_i^{10}}$$

2.5 ความผิดพลาดสำหรับการคำนวณเชิงตัวเลข

ความผิดพลาดที่เกิดขึ้นจากการคำนวณเชิงตัวเลขนั้น มีมาจาก 5 สาเหตุหลักๆ คือ

1) Overflow/Underflow: เกิน/ต่ำกว่าขอบเขตของตัวเลขที่คอมพิวเตอร์กำหนดให้ ดังได้กล่าวแล้วในหัวข้อการแทนค่าระบบจำนวนเต็มและจำนวนจริง
   - บวกเพิ่มตัวเลข 2 จำนวน แล้วให้ผลลัพธ์ที่มีขนาดเกินขอบเขต
   - การคูณตัวเลข 2 จำนวน แล้วให้ผลลัพธ์ที่มีขนาดเกินขอบเขต
   - การหารตัวเลข 2 จำนวน แล้ให้ผลลัพธ์ที่มีทศนยมต่ำกว่าขอบเขต
2) Truncation error: การตัดทศนิยมทิ้ง
3) Rounded-off error: การปัดทศนิยมขึ้น

สำหรับกรณีที่ 1-3 นี้สามารถหลีกเลี่ยงการเกิดความคลาดเคลื่อนที่เกิดจากround-off errors และ overflow/underflow errors ได้

ตัวอย่างที่ 2.7 พิจารณาตัวอย่างสมการต่อไปนี้

$$\frac{xy}{z} = x\left(\frac{y}{z}\right) = \left(\frac{x}{z}\right)y$$

- $\frac{xy}{z}$ ซึ่ง x คูณกับ y นั้นให้ผลคูณต่างกันมาก อาจทำให้เกิด overflow
- $x(\frac{y}{z})$ ซึ่ง y หาร z นั้นให้ผลหารใกล้เคียงกัน จะแทนได้ หรือ
- $\left(\frac{x}{z}\right)y$ ซึ่ง x หาร z นั้นให้ผลหารใกล้เคียงกัน จะแทนได้

ตัวอย่างที่ 2.8 พิจารณาตัวอย่างสมการต่อไปนี้

$$\frac{y^n}{e^{nx}} = \left(\frac{y}{e^x}\right)^n$$

```python
#Lab2_01
import math

x=36
y=10**16
print('x=%d,y=%e'%(x,y))
for n in [-21,-20,-19,19,20]:
    r=(y**n)/math.expm1(n*x)
    print('y**%2d/e**%2dx=%25.15e'%(n,n,r))
    print('(y/e**x)**%2d=%25.15e'%(n,(y/math.expm1(x))**n))
```

```
x=36,y=1.000000e+16
y**-21/e**-21x=   -0.000000000000000e+00
(y/e**x)**-21=    2.121428108447234e-08
y**-20/e**-20x=  -9.999888671826830e-321
(y/e**x)**-20=    4.920700930263791e-08
y**-19/e**-19x=  -1.000000000000000e-304
(y/e**x)**-19=    1.141367814854763e-07
y**19/e**19x=    8.761417546430839e+06
(y/e**x)**19=    8.761417546430882e+06

---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call last)
<ipython-input-80-8093eb639406> in <module>()
      5 print('x=%d,y=%e'%(x,y))
      6 for n in [-21,-20,-19,19,20,21]:
----> 7     r=(y**n)/math.expm1(n*x)
      8     print('y**%2d/e**%2dx=%25.15e'%(n,n,r))
      9     print('(y/e**x)**%2d=%25.15e'%(n,(y/math.expm1(x))**n))

OverflowError: math range error
```

ตัวอย่างที่ 2.9 พิจารณาตัวอย่างสมการต่อไปนี้

$$f_1(x) = \frac{1-\cos x}{x^2}, \qquad f_2(x) = \frac{\sin^2 x}{x^2(1+\cos x)}$$

- กรณีที่ $x \approx \pi$ จะทำให้สมการ $f_2(x)$ เกิด bad subtraction เพราะ $\cos(\pi) = \cos(180) = -1$ ดังนั้นตัวหารจะเป็น 0 จึงควรหลีกเลี่ยงมาใช้ $f_1(x)$
- กรณีที่ $x \approx 0$ จะทำให้สมการ $f_1(x)$ เกิด bad subtraction เพราะ $1 - \cos(0) = 1$ ดังนั้นตัวตั้งจะเป็น 0 จึงควรหลีกเลี่ยงมาใช้ $f_2(x)$

```python
#Lab2_02 round-off error test
import math
for k in [0,1]:
    x=k*math.pi
    tmp=1
    for k1 in range(1,9):
        tmp=tmp*0.1
        x1=x+tmp
        f1=(1-math.cos(x1))/(x1**2)
        f2=((math.sin(x1))**2)/((x1**2)*(1+math.cos(x1)))
        print('At x = %10.8f,\tf1(x) = %18.12e,\t f2(x) = %18.12e'
              %(x1,f1,f2))
```

```
At x = 0.10000000, f1(x) = 4.995834721974e-01, f2(x) = 4.995834721974e-01
At x = 0.01000000, f1(x) = 4.999958333474e-01, f2(x) = 4.999958333472e-01
At x = 0.00100000, f1(x) = 4.999999583255e-01, f2(x) = 4.999999583333e-01
At x = 0.00010000, f1(x) = 4.999999969613e-01, f2(x) = 4.999999995833e-01
At x = 0.00001000, f1(x) = 5.000000413702e-01, f2(x) = 4.999999999958e-01
At x = 0.00000100, f1(x) = 5.000444502912e-01, f2(x) = 5.000000000000e-01
At x = 0.00000010, f1(x) = 4.996003610813e-01, f2(x) = 5.000000000000e-01
At x = 0.00000001, f1(x) = 0.000000000000e+00, f2(x) = 5.000000000000e-01
At x = 3.24159265, f1(x) = 1.898571371550e-01, f2(x) = 1.898571371550e-01
At x = 3.15159265, f1(x) = 2.013534055392e-01, f2(x) = 2.013534055391e-01
At x = 3.14259265, f1(x) = 2.025133720884e-01, f2(x) = 2.025133720914e-01
At x = 3.14169265, f1(x) = 2.026294667803e-01, f2(x) = 2.026294678432e-01
At x = 3.14160265, f1(x) = 2.026410772244e-01, f2(x) = 2.026410604538e-01
At x = 3.14159365, f1(x) = 2.026422382785e-01, f2(x) = 2.026242248740e-01
At x = 3.14159275, f1(x) = 2.026423543841e-01, f2(x) = 2.028044503269e-01
```

```
ZeroDivisionError                         Traceback (most recent call l
<ipython-input-96-9d01c8d90167> in <module>()
      8         x1=x+tmp
      9         f1=(1-math.cos(x1))/(x1**2)
---> 10         f2=((math.sin(x1))**2)/((x1**2)*(1+math.cos(x1)))
     11         print('At x = %10.8f, f1(x) = %18.12e, f2(x) = %18.12e'
     12               %(x1,f1,f2))

ZeroDivisionError: float division by zero
```

สำหรับการแก้ปัญหาการเกิดเหตุการณ์ดังกล่าวนี้ สามารถทำได้โดยใช้อนุกรมเทเลอร์ (Taylor series) ซึ่งจะแก้ปัญหาการเกิด Truncation error ซึ่งจะได้กล่าวถึงภายหลัง

4) Loss of Significance: เกิดจากการลบเลขที่มีค่าเท่ากันหรือใกล้เคียงกัน เรียกว่า bad subtraction

ตัวอย่างที่ 2.10 กรณีลบค่าที่ใกล้เคียงกันที่เกิดจากการคำนวณ ดังสมการต่อไปนี้

$$f_1(x) = \sqrt{x}(\sqrt{x+1} - \sqrt{x}), \qquad f_2(x) = \frac{\sqrt{x}}{\sqrt{x+1} + \sqrt{x}}$$

กำหนดให้ $x = 10^{15} \approx 2^{52}$

$$\sqrt{x+1} = 3.162277660168381 \times 10^7 = 31622776.60168381$$

$$\sqrt{x} = 3.162277660168379 \times 10^7 = 31622776.60168379$$

$$\sqrt{x+1} + \sqrt{x} = 63245553.20336761$$

$$\sqrt{x+1} - \sqrt{x} = 0.000000018626451492309570 \approx 0.00000002$$

```python
#Lab2_03
import math
x=1.0
for k in range(15):
    f1=math.sqrt(x)*(math.sqrt(x+1)-math.sqrt(x))
    f2=math.sqrt(x)/(math.sqrt(x+1)+math.sqrt(x))
    print('At x=%15.0f,  f1(x)=%25.23f,  f2(x)=%20.23f'
          %(x,f1,f2))
    x=10*x
sx1=math.sqrt(x+1)
sx=math.sqrt(x)
d=sx1-sx
s=sx1+sx
print('sqrt(x+1)=%25.13f,\tsqrt(x)=%25.13f'%(sx1,sx))
print('diff=%25.23f,\tsum=%25.23f'%(d,s))
```

```
At x=              1,  f1(x)=0.41421356237309514547462,  f2(x)=0.41421356237309508996347
At x=             10,  f1(x)=0.48808848170151475365230,  f2(x)=0.48808848170151547529727
At x=            100,  f1(x)=0.49875621120889945814270,  f2(x)=0.49875621120890273330062
At x=           1000,  f1(x)=0.49987506246102186846514,  f2(x)=0.49987506246096485851282
At x=          10000,  f1(x)=0.49998750062485441958415,  f2(x)=0.49998750062496088997221
At x=         100000,  f1(x)=0.49999875000592886031825,  f2(x)=0.49999875000624993681697
At x=        1000000,  f1(x)=0.49999987504634191282094,  f2(x)=0.49999987500006248808404
At x=       10000000,  f1(x)=0.49999998740115092488168,  f2(x)=0.49999998750000057556875
At x=      100000000,  f1(x)=0.50000000555883161723614,  f2(x)=0.49999999874999995208569
At x=     1000000000,  f1(x)=0.50000007799750634251978,  f2(x)=0.49999999987499998965745
At x=    10000000000,  f1(x)=0.49999944167211651802063,  f2(x)=0.49999999998750005447690
At x=   100000000000,  f1(x)=0.50000444963116807972625,  f2(x)=0.49999999999874999989657
At x=  1000000000000,  f1(x)=0.5000380724668502807617,  f2(x)=0.49999999999987498888743
At x= 10000000000000,  f1(x)=0.49919454697383597308047,  f2(x)=0.49999999999998750999097
At x=100000000000000,  f1(x)=0.50291419029235839843750,  f2(x)=0.49999999999999872324352
sqrt(x+1)=   31622776.6016838103533,    sqrt(x)=   31622776.6016837917268
diff=0.00000001862645149230957, sum=63245553.20336760580539703369141
```

5) Numerical algorithms: การเขียนอัลกอริทึมทางการคำนวณที่ผิดพลาด ซึ่งเป็นความผิดพลาดที่เกิดขึ้นจากมนุษย์นั่นเอง ดังนั้นโปรแกรมเมอร์จำเป็นต้องศึกษาวิธีการเขียนโปรแกรมที่ดี

Assignment Week2

เขียนโปรแกรมส่งในชั่วโมง

1. $\sum_{i=1}^{N} i$ เมื่อ i เป็นเลขคี่ ให้เขียนโดยใช้การทำซ้ำ ห้ามใช้ฟังก์ชัน sum
   1.1.เมื่อ $N = 50$,
   1.2.เมื่อ $N = 10^7$
   1.3.เมื่อ $N = 10^9$
2. $\sum_{i=1}^{N} i$ เมื่อ i เป็นเลขคู่ ให้เขียนโดยใช้การทำซ้ำ ห้ามใช้ฟังก์ชัน sum
   2.1.เมื่อ $N = 50$,
   2.2.เมื่อ $N = 10^7$
   2.3.เมื่อ $N = 10^9$
3. ถ้ากำหนด $x = 9.8^{201}$, $y = 10.2^{199}$ จงหาวิธีหลีกเลี่ยงการเกิด error, overflow, underflow ของสมการต่อไปนี้
   3.1.$z = \sqrt{x^2 + y^2}$
   3.2.$z = y\sqrt{(\frac{x}{y})^2 + 1}$
4. จง plot ค่า x จำนวน 100 ค่า ที่อยู่ระหว่าง $[10^{14}, 10^{16}]$ ลงบน graph โดยคำนวณค่า x และ y จากสมการต่อไปนี้
   4.1.$y = \sqrt{2x^2 + 1} - 1$
   4.2.$y = \frac{2x^2}{\sqrt{2x^2+1}+1}$

และสมการใดที่ช่วยแก้ปัญหาการเกิด loss of significance

2.6 การคำนวณความคาดเคลื่อน (Error)

ความคาดเคลื่อน เป็นผลต่างระหว่างค่าจริงในการคำนวณและค่าประมาณซึ่งถูกใช้ในวิธีการเชิงตัวเลข

$Error\ (e) = true\ value - approximate\ value$ ค่าคลาดเคลื่อน (e) = ค่าจริง − ค่าประมาณ	2.1

ความคลาดเคลื่อนมี 3 ประเภทคือ

1) ความคลาดเคลื่อนสมบูรณ์ (Absolute error: e_{abs})
2) ความคลาดเคลื่อนสัมพัทธ์ (Relative error: e_{rel})
3) ความคลาดเคลื่อนจากการทำซ้ำ (Error of iterative method)

2.6.1 ความคลาดเคลื่อนสมบูรณ์ (Absolute error: e_{abs})

คือขนาดของค่าคลาดเคลื่อนระหว่างค่าจริงกับค่าประมาณซึ่งอยู่ในรูปแบบค่าสัมบูรณ์

$e_{abs} = \lvert true\ value - approx.\ value\rvert$	2.2

2.6.2 ความคลาดเคลื่อนสัมพัทธ์ (Relative error: e_{rel})

คืออัตราส่วนของความคลาดเคลื่อนระหว่างค่าจริงกับค่าประประมาณเมื่อเทียบกับค่าจริง

$e_{rel} = \dfrac{e_{abs}}{\lvert true\ value\rvert}$ หรือ $\varepsilon_t = e_{rel} \times 100\%$	2.3

ε_t เรียกว่า ความคลาดเคลื่อนสัมพัทธ์คิดเป็นร้อยละจริง (True percent relation error) และ $true\ value \neq 0$

ในทางปฏิบัติถ้าหาค่าจริงไม่ได้ ค่าความคลาดเคลื่อนที่คำนวณได้จะเป็นเพียงความคลาดเคลื่อนโดยประมาณ ซึ่งอยู่ในรูปความคลาดเคลื่อนสัมพัทธ์โดยประมาณ (approximation relative error) ดังนี้

$e_a = \dfrac{\lvert approx.\ value_{new} - approx.\ value_{old}\rvert}{\lvert approx.\ value_{new}\rvert}$ หรือ $\varepsilon_a = e_a \times 100\%$	2.4

2.6.3 ความคลาดเคลื่อนจากการทำซ้ำ (Error of iterative method)

เป็นความคลาดเคลื่อนที่เกิดขึ้นจากการทำงานที่มีการวนซ้ำ ใช้ในการทำงานจริงเพื่อกำหนดเกณฑ์การหยุด ซึ่งสามารถหาความคลาดเคลื่อนสัมพัทธ์โดยประมาณจากการวนซ้ำครั้งที่ n เมื่อ $n \geq 0$ ได้ดังนี้

$\varepsilon_a = \frac{\lvert approx.\ value_n - approx.\ value_{n-1} \rvert}{\lvert approx.\ value_n \rvert} \times 100\%$	2.5

โดยที่

$approx.\ value_n$ คือค่าประมาณที่ได้จากการทำซ้ำครั้งที่ n, $approx.\ value_n \neq 0$

$approx.\ value_{n-1}$ คือค่าประมาณที่ได้จากการทำซ้ำครั้งที่ $n-1$

ตัวอย่างที่ 2.11 การคำนวณหาความคลาดเคลื่อนเมื่อ

กำหนดค่าจริง (true value) = 0.4357,

ค่าประมาณ (approx. value) = 0.4364 จงหาค่าต่างๆ ต่อไปนี้

วิธีทำ

- ความคลาดเคลื่อน (Error)

$$Error\ (e) = true\ value - approximate\ value$$

- ความคลาดเคลื่อนสัมบูรณ์ (Absolute error)

$$e_{abs} = \lvert true\ value - approx.\ value \rvert$$

- ความคลาดเคลื่อนสัมพัทธ์ (Relative error)

$$e_{rel} = \frac{e_{abs}}{\lvert true\ value \rvert}$$

- ความคลาดเคลื่อนสัมพัทธ์คิดเป็นร้อยละจริง (True percent relative error)

$$\varepsilon_t = e_{rel} \times 100\%$$

2.7 การกำหนดความแม่นยำ

เป็นการระบบค่าขอบเขตของความคลาดเคลื่อน มี 3 วิธี

1) กำหนดจำนวนตำแหน่งทศนิยม (Decimal Place, D.P.)
   - 0.3999 มีทศนิยมถูกต้อง 4 ตำแหน่ง (4 D.P.)
   - 10.435 มีทศนิยมถูกต้อง 3 ตำแหน่ง (3 D.P.)
2) กำหนดจำนวนเลขนัยสำคัญ (Significant Digit, S.D.) โดยเลขนัยสำคัญนับตั้งแต่เลขซ้ายสุดที่ไม่ใช่ 0 ตัวแรกถึงเลขตัวสุดท้ายที่จำเป็นต้องเขียน
   - $101.4456 = 1.014456 \times 10^2$ มีเลขนัยสำคัญ 7 ตำแหน่ง (7 S.D.)

     $= 1.0144560 \times 10^2$ มีเลขนัยสำคัญ 8 ตำแหน่ง (8 S.D.)
   - 0.01204 มีเลขนัยสำคัญ 4 ตำแหน่ง (4 S.D.)
3) กำหนดค่าขอบเขตของความคลาดเคลื่อน (ε) ส่วนตัวควบคุมการหยุดการคำนวณสำหรับการวนซ้ำ เรียกว่า Tolerance (ε_s) โดยที่

$\lvert\varepsilon_a\rvert < \varepsilon_s$	2.6

ถ้าต้องการให้ผลลัพธ์ถูกต้องอย่างน้อย n ตำแหน่ง (n S.D.) จะได้

$\varepsilon_s = (0.5 \times 10^{2-n})\%$	2.7

ตัวอย่างที่ 2.12 การกำหนดขอบเขตความคลาดเคลื่อนเมื่อกำหนด $x = 0.3464$ เป็นค่าประมาณของค่าจริง X และมีขอบเขตของความคลาดเคลื่อน $(\varepsilon) = 0.0008$ เขียนได้ดังนี้

<u>วิธีทำ</u>

$$X = x \pm \varepsilon \text{ หรือ } x - \varepsilon \le X \le x + \varepsilon$$

$$X = 0.3464 \pm 0.0008 \text{ หรือ}$$

$$0.3464 - 0.0008 \le X \le 0.3464 + 0.0008$$

ตัวอย่างที่ 2.13 การกำหนดขอบเขตความคลาดเคลื่อนเมื่อกำหนดให้ $x = 10.25175$ ซึ่งมีความถูกต้องที่จำนวนตำแหน่งทศนิยม 5 ตำแหน่ง (D.P.) และมีจำนวนเลขนัยสำคัญ 7 ตำแหน่ง (S.D.) ต้องการให้ค่า x มีความถูกต้องที่ระดับ 1 D.P., 2 D.P., 3 D.P. และ 4 D.P. จงหาค่าขอบเขตของความคลาดเคลื่อน

วิธีทำ

ความถูกต้องที่ระดับ	ค่าประมาณ (x)	ความคลาดเคลื่อน
1 D.P.	10.3	0.4825×10^{-1}
2 D.P.	10.25	0.175×10^{-2}
3 D.P.	10.252	-0.25×10^{-3}
4 D.P.	10.2518	-0.5×10^{-4}

จากตัวอย่างข้างต้น ถ้าจำนวนใดๆ กำหนดการปัดเศษให้เหลือความถูกต้องที่ n D.P. ซึ่งค่าขอบเขตของความคลาดเคลื่อนจะได้ 0.5×10^{-n} ดังนั้น

$$e_{abs} = |true\ value - approx.\ value| \leq 0.5 \times 10^{-n}$$

ตัวอย่างที่ 2.14 กำหนดให้ $x = 10.25$ ซึ่งมีความถูกต้องที่จำนวนตำแหน่งทศนิยม 2 ตำแหน่ง (D.P.) ขนาดของความคลาดเคลื่อนสูงสุดคือ $\varepsilon = 0.5 \times 10^{-n}$

วิธีทำ

$$X = 10.25 + 0.005 \text{ หรือ } 10.245 \leq X \leq 10.255$$

พิจารณาจำนวนเลขนัยสำคัญ $x = 10.25$ มีจำนวนเลขนัยสำคัญ 4 ตำแหน่ง (S.D.) กำหนดความถูกต้องที่จำนวนตำแหน่งทศนิยม 2 ตำแหน่ง

หาค่าขอบเขตของความคลาดเคลื่อนสัมพัทธ์ได้ดังนี้

$$e_{rel} = \frac{e_{abs}}{|true\ value|} = \frac{0.5 \times 10^{-2}}{10.25} = 0.4878 \times 10^{-3} = 4.878 \times 10^{-4}$$

$$\therefore \text{ จะได้ว่า } 4.878 \times 10^{-4} \leq 0.5 \times 10^{-2}$$

แบบฝึกหัด (ความคลาดเคลื่อน)

1. กำหนดให้ ค่าจริง 10,000 ค่าประมาณ 9,999 จงคำนวณหาค่าความคลาดเคลื่อนและค่าความคลาดเคลื่อนสัมพัทธ์คิดเป็นร้อยละจริง
2. จงคำนวณหาค่าความคลาดเคลื่อนสัมบูรณ์และค่าความคลาดเคลื่อนสัมพัทธ์จากข้อย่อยต่อไปนี้

   2.1.ค่าจริง = π ค่าประมาณ $\frac{22}{7}$

   2.2.ค่าจริง = π ค่าประมาณ 3.1416

   2.3.ค่าจริง = e ค่าประมาณ 2.718

   2.4.ค่าจริง = $\sqrt{2}$ ค่าประมาณ 1.414

2.5.ค่าจริง = e^{10} ค่าประมาณ 22,000

2.6.ค่าจริง = 10^{π} ค่าประมาณ 1,400

2.7.ค่าจริง = 8! ค่าประมาณ = 39,900

3. จงคำนวณหาค่าความคลาดเคลื่อนสัมบูรณ์และค่าความคลาดเคลื่อนสัมพัทธ์โดยกำหนดความถูกต้องที่จำนวนเลขนัยสำคัญอย่างน้อย 5 ตำแหน่ง (S.D) จากข้อย่อยต่อไปนี้

3.1.ค่าจริง = $133 + 0.921$ ค่าประมาณ = 134

3.2.ค่าจริง = $133 - 0.499$ ค่าประมาณ = 133

3.3.ค่าจริง = $(121 - 0.327) - 119$ ค่าประมาณ = 2.00

3.4.ค่าจริง = $(121 - 119) - 0.327$ ค่าประมาณ = 1.67

3.5.ค่าจริง = $(\frac{2}{9})(\frac{9}{7})$ ค่าประมาณ = 0.287

3.6.ค่าจริง = $-10\pi + 6e - \frac{3}{62}$ ค่าประมาณ = -15.1

2.8 การเขียนโปรแกรมที่ดี

1) การเขียนโปรแกรมที่ใช้ความสัมพันธ์แบบโครงสร้างต่อเนื่อง (Nested computing)

โดยการใช้ความสัมพันธ์แบบเวียนซ้ำ (Recurrent relation) ทั้งนี้เพื่อเพิ่มประสิทธิภาพในเรื่องเวลาในการประมวลผล (time-efficient computation)

2) การใช้ตัวดำเนินการ vector กับ loop iteration

ซึ่งการเขียนโปรแกรมทำงานกับ vector จะสามารถทำงานได้เร็วกว่า loop iteration

3) Iterative กับ Nested routine

กรณี nested routine ที่เป็น recursive ต้องระวัง ควรพิจารณาอย่างเหมาะสมเลือกใช้ให้เหมาะกับงาน ทั้งนี้ควรพิจารณา

- Runtime error เพราะ stack memory เต็มเนื่องจากจำนวนรอบของการเรียกตัวเองมีจำนวนมาก
- เวลาที่ใช้สำหรับจำนวนของการเรียกใช้ฟังก์ชัน (n)

4) หลีกเลี่ยงปัญหาการเกิด Runtime error

5) การใช้ตัวแปรโกลบอล (Global variable)

การประกาศเป็นตัวแปรแบบโกลบอล นิยมใช้สำหรับตัวแปรที่เป็นค่าคงที่ และในการประกาศตัวแปรโกลบอล พบว่า

- ข้อดี: ไม่ยุ่งยากซับซ้อนในการส่งผ่านค่า
- ข้อเสีย:
- ไม่สะดวกในการใช้งาน

- อาจมีการใช้ตัวแปรซ้ำซ้อนภายในโลคอล (Local) หรือฟังก์ชัน (Function) และทำให้เกิดความสับสนต่อผู้ใช้เองว่าต้องการอ้างภายในโลคอล หรือภายนอก หรือความสัมพันธ์ของตัวแปรดังกล่าว
- ค่าที่ใช้อาจถูกเปลี่ยนแปลงค่าอัตโนมัติ โดยความสับสนของผู้ใช้
- เปลืองเนื้อที่หน่วยความจำ

6) ควรให้มีการส่งผ่านค่าสำหรับฟังก์ชันได้หลากหลาย

ตัวอย่างที่ 2.15 การเขียนโปรแกรมโดยใช้ vector เปรียบเทียบกับ loop iteration (recurrent relation) ให้เป็น เช่นการหาผลลัพธ์ของสมการพหุนาม (Polynomial)

$$p_4\ (x)\ =\ a_1x^4\ + a_2x^3 + a_3x^2 + a_4x + a_5$$

จัดรูปสมการใหม่ให้อยู่ในรูปของ nested structure

$$p_{4n}\ (x) = \Big(\big((a_1x + a_2)x + a_3\big)x + a_4\Big)\,x + a_5$$

จากตัวอย่างข้างต้น สามารถนำมาโปรแกรมได้โดย

$$\sum_{i=1}^{N-1} a_i x^i$$

```python
# Lab2_04 (Polynomial)
import numpy as np
import time

N=1000000
a=np.arange(1,N+1,dtype=np.float32)
x=1
print('plain multiplication')
start_time=time.time()
p=np.sum(a*(x**np.arange(N-1,-1,-1,dtype=np.int)),dtype=np.float64)
print('p=%f'%p)
stop_time=time.time()
print('time=',stop_time-start_time)

print('\nnested multiplication')
start_time=time.time()
pn=a[0]
for i in range(1,N):
    pn=pn*x+a[i]
print('p=%f'%pn)
stop_time=time.time()
print('time=',stop_time-start_time)

print('\nfunction polynomial')
start_time=time.time()
pp=np.polyval(a,x)
print('p=%f'%pp)
stop_time=time.time()
print('time=',stop_time-start_time)
```

ตัวอย่างที่ 2.16 การเขียนโปรแกรมโดยใช้ Nested structure (Recurrent relation) ซึ่งเป็นความสัมพันธ์แบบเวียนซ้ำ การหาการแจกแจงความน่าจะเป็นปัวส์ซอง (Poisson probability distribution) สมมติ $\lambda = 100$ และ $\mathrm{k} = 155$

$$F(K) = \sum_{k=0}^{K} \frac{\lambda^k}{k!} e^{-\lambda}$$

```python
#lab2_05_1: Poisson (Nested Structure)
import numpy as np

lam,K=100,155
p=np.exp(-lam)
F=0
for k in range(1,K):
    p*=lam/k
    F+=p
print('F=',F)
print('F=%.15f'%F)
```

```python
#lab2_05_2: Poisson (Non-nested Structure)
import numpy as np
import math

lam,K=100,155
F=0
for k in range(1,K):
    p=lam**k/math.factorial(k)
    F+=p
print('F=',(F*np.exp(-lam)))
print('F=%.15f'%(F*np.exp(-lam)))
```

ตัวอย่างที่ 2.17 การเขียนโปรแกรมแบบ Recursive และ Iterative

```python
#lab2_06_1:factorial
# iterative
def fact1(n):
    k=1
    for i in range(2,n+1):
        k=k*i
    return k

# recursive
def fact2(n):
    if n<=1:
        k=1
    else:
        k=n*fact2(n-1)
    return k
n=5
print('fact1: %d!=%d'%(n,fact1(n)))
print('fact2: %d!=%d'%(n,fact2(n)))
```

3 พีชคณิตเมทริกซ์ (Matrix Algebra)

3.1 เมทริกซ์และการดำเนินการบนเมทริกซ์ (Matrix and Operation)

3.1.1 นิยามของเมทริกซ์ (Matrix Definition)

นิยามที่ 3.1 เมทริกซ์ (Matrix) คือ ชุดข้อมูลที่มีการเรียงกันเป็นแนวแถว (Row) และแนวหลัก (Column) เขียนอยู่ภายใต้ [ ] หรือ ( )

- นิยมใช้ตัวพิมพ์ใหญ่ $(A, B, C, \ldots)$ แทนเมตริกซ์
- สมาชิกของเมทริกซ์จะแทนด้วยตัวพิมพ์เล็กพร้อมทั้งระบุตำแหน่งของสมาชิกว่าอยู่ตำแหน่งใดของเมทริกซ์ เช่น
    - a_{12} แทนสมาชิกในแถวที่ 1 หลักที่ 2 ของเมทริกซ์ A
    - b_{22} แทนสมาชิกในแถวที่ 2 หลักที่ 2 ของเมทริกซ์ B
- มีมิติเป็น $(m \times n)$, $m =$จำนวนแถว และ $n =$จำนวนหลัก

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

ตัวอย่างที่ 3.1: จงหาขนาดของเมทริกซ์

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 3 & 4 \end{bmatrix}, \qquad C = \begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix}$$

<u>วิธีทำ</u> A มีขนาด 3x2 B มีขนาด 1x3 และ C มีขนาด 3x1

นิยามที่ 3.2 เมทริกซ์ศูนย์ (Zero Matrix) คือ เมทริกซ์ที่มีสมาชิกทุกตัวเป็นศูนย์ เขียนแทนด้วยสัญลักษณ์ $\underline{0}$

ตัวอย่างที่ 3.2: จงหาขนาดของเมทริกซ์

$$X = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad Y = \begin{bmatrix} 0 & 0 \end{bmatrix}, \qquad Z = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

<u>วิธีทำ</u> $X = \underline{0}$ เป็นเมทริกซ์ขนาด 3x3

$Y = \underline{0}$ เป็นเมทริกซ์ขนาด 1x2

$Z = \underline{0}$ เป็นเมทริกซ์ขนาด 3x1

นิยามที่ 3.3 เมทริกซ์จัตุรัส (Square Matrix) หมายถึงเมทริกซ์ที่มีจำนวนแถวเท่ากับจำนวนหลัก มีมิติเป็น $n \times n$ เขียนแทนด้วย A_n

ตัวอย่างที่ 3.3:ตัวอย่างเมทริกซ์จตุรัส

$$A_2 = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}, B_3 = \begin{bmatrix} 3 & 2 & 1 \\ 5 & 1 & 2 \\ 4 & 5 & 6 \end{bmatrix}$$

นิยามที่ 3.4 เมทริกซ์แนวทแยง (Diagonal Matrix) หมายถึงเมทริกซ์จตุรัส (A_n)ที่มีสมาชิกในแนวทแยงเป็นศูนย์ $(a_{ij} = 0)$ สำหรับทุกๆ ค่า $i = j$

นิยามที่ 3.5 เมทริกซ์สามเหลี่ยม (Triangular Matrix) หมายถึงเมทริกซ์จตุรัส (A_n)ที่มีสมาชิกทุกตัวที่อยู่เหนือหรือใต้แนวทแยงเป็นศูนย์ทั้งหมด ดังนั้น

เมทริกซ์สามเหลี่ยมล่าง (Lower Triangular Matrix) หมายถึงเมทริกซ์จตุรัส (A_n)ที่มีสมาชิกทุกตัวที่อยู่เหนือแนวทแยงเป็นศูนย์ทั้งหมด $(a_{ij} = 0)$ สำหรับทุกๆ ค่า $i < j$

เมทริกซ์สามเหลี่ยมบน (Upper Triangular Matrix) หมายถึงเมทริกซ์จตุรัส (A_n)ที่มีสมาชิกทุกตัวที่อยู่ใต้แนวทแยงเป็นศูนย์ทั้งหมด $(a_{ij} = 0)$ สำหรับทุกๆ ค่า $i > j$

ตัวอย่างที่ 3.4: กำหนดให้ $A_3 = \begin{bmatrix} 0 & 2 & 1 \\ 5 & 0 & 2 \\ 4 & 5 & 0 \end{bmatrix}$, $B_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 4 & 3 & 0 & 0 \\ 2 & 1 & 1 & 0 \\ 1 & 1 & 5 & 4 \end{bmatrix}$ และ

$C_3 = \begin{bmatrix} 5 & 2 & 1 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{bmatrix}$ เป็นเมทริกซ์ประเภทใด

วิธีทำ A_3 เป็นเมทริกซ์แนวทแยง, B_4 เป็นเมทริกซ์สามเหลี่ยมล่าง และ C_4 เป็นเมทริกซ์สามเหลี่ยมบน,

นิยามที่ 3.6 เมทริกซ์เอกลักษณ์ (Identity Matrix) หมายถึงเมทริกซ์จตุรัส (A_n)ที่มีสมาชิกในแนวทแยงมีค่าเป็นหนึ่ง $(a_{ij} = 1)$ สำหรับทุกๆ ค่า $i = j$ นอกนั้นเป็นศูนย์ $(a_{ij} = 0)$ สำหรับทุกๆ ค่า $i \neq j$ เขียนแทนด้วย I_n

ตัวอย่างที่ 3.5:ตัวอย่างเมทริกซ์เอกลักษณ์

$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, B_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

นิยามที่ 3.7 เมทริกซ์สมมาตร (Symmetric Matrix) คือเมทริกซ์ A ที่เป็นเมทริกซ์จตุรัสแล้ว $A^t = A$ นั่นคือ ถ้า $a_{ij} = a_{ji}$ สำหรับทุกค่า i และ j

ตัวอย่างที่ 3.6: กำหนดให้ $A = \begin{bmatrix} 1 & -2 & 4 \\ -2 & 2 & 3 \\ 4 & 3 & 5 \end{bmatrix}$ จงพิจารณาว่า A เป็นเมทริกซ์สมมาตรหรือไม่

วิธีทำ

นิยามที่ 3.8 เมทริกซ์เสมือนสมมาตร (Skew Symmetric Matrix) คือเมทริกซ์ A ที่เป็นเมทริกซ์จตุรัส แล้ว $A^t = -A$ นั่นคือ ถ้า $a_{ij} = -a_{ji}$ สำหรับทุกค่า i และ j

ตัวอย่างที่ 3.7: กำหนดให้ $A = \begin{bmatrix} 1 & -3 & 4 \\ 3 & 2 & -2 \\ -4 & 2 & 0 \end{bmatrix}$ จงพิจารณาว่า A เป็นเมทริกซ์เสมือนสมมาตรหรือไม่

วิธีทำ

3.1.2 การดำเนินการบนเมทริกซ์ (Matrix Operating Rule)

นิยามที่ 3.9 การเท่ากันของเมทริกซ์ (Equality of Matrices)
ถ้า $A = [a_{ij}]_{m\times n}, B = [b_{ij}]_{p\times q}$ แล้ว $A = B$ ก็ต่อเมื่อ $m = p, n = q$ และ $a_{ij} = b_{ij}$ สำหรับทุกค่า i, j

ตัวอย่างที่ 3.8: กำหนดให้ $A = \begin{bmatrix} 1 & 2 \\ a & 4 \\ 5 & 6 \end{bmatrix}$ และ $B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & b \end{bmatrix}$

ดังนั้น $A = B$ ก็ต่อเมื่อ $a = 3$ และ $b = 6$

นิยามที่ 3.10 การบวกและลบเมทริกซ์ (Matrix Addition and Subtraction)
ถ้า $A = [a_{ij}]_{m\times n}, B = [b_{ij}]_{m\times n}$ แล้ว $A + B = [a_{ij} + b_{ij}]$ และ $A - B = [a_{ij} - b_{ij}]$

ตัวอย่างที่ 3.9: ให้ $A = \begin{bmatrix} 1 & 1 & 2 \\ 3 & 2 & 0 \end{bmatrix}, B = \begin{bmatrix} 5 & 3 & 2 \\ 1 & 2 & 3 \end{bmatrix}$

$$\therefore A + B = \begin{bmatrix} 1 & 1 & 2 \\ 3 & 2 & 0 \end{bmatrix} + \begin{bmatrix} 5 & 3 & 2 \\ 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 1+5 & & \\ & & \end{bmatrix}$$

$$= \begin{bmatrix} 6 & & \\ & & \end{bmatrix}$$

$$\therefore A - B = \begin{bmatrix} 1 & 1 & 2 \\ 3 & 2 & 0 \end{bmatrix} - \begin{bmatrix} 5 & 3 & 2 \\ 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} & & \\ & & \end{bmatrix}$$

$$= \begin{bmatrix} 6 & & \\ & & \end{bmatrix}$$

นิยามที่ 3.11 การคูณเมทริกซ์ด้วยค่าคงที่หรือสเกลาร์ (Scalar Multiplication Matrix)

ถ้า $A = [a_{ij}]_{m \times n}$ และ c เป็นค่าคงที่หรือสเกลาร์แล้ว จะได้ว่า

$$cA = [ca_{ij}]_{m \times n}$$

ตัวอย่างที่ 3.10: กำหนดให้ $A = \begin{bmatrix} 1 & 2 \\ 3 & 0 \\ 5 & 1 \end{bmatrix}$ ดังนั้น $3A = \begin{bmatrix} 3 & 6 \\ 9 & 0 \\ 15 & 3 \end{bmatrix}$

ตัวอย่างที่ 3.11: จงหาค่า a และ b เมื่อกำหนดเงื่อนไขต่อไปนี้

$$\begin{bmatrix} a^2 & 6 \\ b^2 & -7 \end{bmatrix} = \begin{bmatrix} 25 & 4 + b \\ 4 & a - 2 \end{bmatrix}$$

จะได้ว่า $a^2 = 25$ และ $a - 2 = -7$ นั่นคือ

$a =$ ____ และ $a =$ ___ ดังนั้น $a =$ _____

และจาก $b^2 = 4$ และ $4 + b = 6$ นั่นคือ

$b =$ ____ และ $b =$____ ดังนั้น $b =$____

นิยามที่ 3.12 การคูณเมทริกซ์ด้วยเมทริกซ์ (Matrix Multiplication)

ถ้า $A = [a_{ij}]_{m \times n}, B = [b_{ij}]_{n \times r}$ แล้ว ถ้า $C = AB$ แล้ว

$$C = [c_{ij}]_{m \times r}$$

โดยที่

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{in} b_{nj}$$

ตัวอย่างที่ 3.12: ให้ $A = \begin{bmatrix} 1 & 1 & 2 \\ 3 & 2 & 0 \end{bmatrix}, B = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}$

$$\therefore AB = \begin{bmatrix} 1 & 1 & 2 \\ 3 & 2 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 1(1) + 1(0) + 2(2) \\ 3(1) + 2(0) + 0(2) \end{bmatrix} =$$

3.2 คุณสมบัติของเมตริกซ์ (Matrix Properties)

3.2.1 คุณสมบัติการบวกของเมตริกซ์

กำหนดให้ A, B และ C เป็นเมตริกซ์ขนาด $m \times n$

- การเปลี่ยนกลุ่ม

$$(A + B) + C = A + (B + C)$$

- การสลับที่

$$A + B = B + A$$

- เมทริกซ์เอกลักษณ์การบวก $\underline{0}$ ที่ทำให้

$$A + \underline{0} = A = \underline{0} + A$$

- A มีอินเวอร์สเมทริกซ์การบวก $-A$ ที่ทำให้

$$A + (-A) = \underline{0} = (-A) + A$$

3.2.2 คุณสมบัติการคูณเมตริกซ์ด้วยเมตริกซ์

กำหนดให้ A, B และ C เป็นเมตริกซ์ขนาด $n \times n$

- การเปลี่ยนกลุ่ม

$$(AB)C = A(BC)$$

- เมทริกซ์เอกลักษณ์การคูณ I ขนาด $n \times n$ ที่ทำให้ $AI = A = IA$

โดยที่ $I_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$

3.2.3 คุณสมบัติการคูณเมทริกซ์ด้วยค่าคงที่หรือสเกลาร์

กำหนดให้ A, B เป็นเมทริกซ์ขนาด $m \times n$ และ c, d เป็นสเกลาร์

- การเปลี่ยนกลุ่ม

$$(cd)A = c(dA) = (cA)d$$

- เอกลักษณ์การคูณ

$$1A = A$$

- การกระจาย

$$c(A+B) = cA + cB \text{ และ } (c+d)A = cA + dA$$

ตัวอย่างที่ 3.13: กำหนด $A = \begin{bmatrix} 1 & -2 \\ 0 & 3 \end{bmatrix}, B = \begin{bmatrix} -3 & 4 \\ 2 & 1 \end{bmatrix}$ จงหา X เมื่อ

$2X + A = B$

$$2X = B - A = \begin{bmatrix} & \\ & \end{bmatrix} - \begin{bmatrix} & \\ & \end{bmatrix} = \begin{bmatrix} & \\ & \end{bmatrix}$$

$X =$ =

3.3 ทรานสโพสของเมทริกซ์ (Transpose of matrix)

นิยามที่ 3.13 ถ้า $A = [a_{ij}]_{m \times n}$ ทรานสโพสของ A แทนด้วย A^t, A^T, A' โดยที่

$$A^t = [a_{ji}]_{n \times m}$$

ตัวอย่างที่ 3.14: $A = \begin{bmatrix} 1 & 1 & 2 \\ 3 & 2 & 0 \end{bmatrix}$ จะได้ $A^t = \begin{bmatrix} 1 & 3 \\ 1 & 2 \\ 2 & 0 \end{bmatrix}$

นิยามที่3.14 สำหรับเมทริกซ์ A และ C เป็น $m \times n$ และ B เป็น $n \times p$ ที่สามารถบวกหรือคูณกันได้

$$(A^t)^t = A$$

$$(A + C)^t = A^t + C^t$$

$$(AB)^t = B^t A^t$$

$$(A^m)^t = (A^t)^m, \quad \text{โดยที่ } m \in \mathbb{Z}$$

$$(kA)^t = kA^t, \quad \text{โดยที่ } k \in \mathbb{R}$$

ตัวอย่างที่ 3.15: $A = \begin{bmatrix} 1 & 1 & 2 \\ 3 & 2 & 0 \end{bmatrix}$ จะได้ $B = \begin{bmatrix} 2 & 1 & 4 \\ 3 & 5 & 3 \end{bmatrix}$ จงแสดงว่า

$$(A+C)^t = A^t + C^t$$

<u>วิธีทำ</u> $(A+C)^t = \begin{bmatrix} 1+2 & 1+1 & 2+4 \\ 3+3 & 2+5 & 0+3 \end{bmatrix}^t = \begin{bmatrix} 3 & 2 & 6 \\ 6 & 7 & 3 \end{bmatrix}^t = \begin{bmatrix} 3 & 6 \\ 2 & 7 \\ 6 & 3 \end{bmatrix}$

$A^t + C^t =$

3.4 ค่ากำหนด (Determinant)

Determinant ของ matrix เป็นฟังก์ชันที่มี domain เป็นเซตของเมทริกซ์จัตุรัส และมี range เป็นสับเซตของจำนวนจริง โดยแทนด้วยสัญลักษณ์ $\det(A)$ หรือ $|A|$

นิยามที่ 3.15 กำหนดให้ A เป็นเมทริกซ์ $n \times n$

ถ้า $\det(A) = 0$ เรียก A ว่าเป็นเมทริกซ์เอกฐาน (Singular matrix)

ถ้า $\det(A) \neq 0$ เรียก A ว่าเป็นเมทริกซ์ไม่เป็นเอกฐาน (Non-singular matrix)

นิยามที่ 3.16

- ถ้า A เป็น Non-singular matrix แล้ว A สามารถหาอินเวอร์สการคูณได้
- ถ้า $A = [a]$ เป็นเมตริกซ์มิติ $1 \times 1, a$ เป็นจำนวนจริง แล้ว $det\ (A) = a$
- ถ้า $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ เป็นเมตริกซ์มิติ $2 \times 2, a, b, c$ และ d เป็นจำนวนจริงแล้ว

$$det\ (A) = ad - bc$$

ตัวอย่างที่ 3.16: จงหา $det\ (A)$

1. $A = [3]$ จะได้ว่า $\det(A) = 3$
2. $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ จะได้ว่า $\det(A) = ad - bc = 1(4) - 2(3) = -2$

นิยามที่ 3.17 กำหนดให้ $A = [a_{ij}]$ เป็นเมตริกซ์ $n \times n$ และ c_{ij} เป็น co-factor ใน ตำแหน่งของแถวที่ i และหลักที่ j, $(i, j = 1,2,3, \dots, n)$ แล้ว

$$\det(A) = \sum_{i=1}^{n} c_{ij} a_{ij} \text{ สำหรับ } j = 1,2,3, \dots, n \text{ หรือ}$$

$$= \sum_{j=1}^{n} c_{ij} a_{ij} \text{ สำหรับ } i = 1,2,3, \dots, n$$

โดยที่โคแฟคเตอร์ (co-factor) ในตำแหน่งที่อยู่ในแถวที่ i หลักที่ j เขียนแทนด้วย

$$\boldsymbol{c_{ij}} = (-\boldsymbol{1})^{\boldsymbol{i+j}} \times \det$$

ของเมทริกซ์ย่อย (Sub-matrix) ที่เกิดจากการ ตัดแถวที่ i และหลักที่ j ออกจากเมทริกซ์นั้น

ตัวอย่างที่ 3.17: จงหา $\boldsymbol{det(A)}$

ถ้าให้ $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix}$

$c_{11} = (-1)^{(1+1)} \begin{vmatrix} 5 & 6 \\ 8 & 10 \end{vmatrix} = (-1)^2\big(5(10) - 6(8)\big) = 2,$

$c_{12} = (-1)^{(1+2)} \begin{vmatrix} 4 & 6 \\ 7 & 10 \end{vmatrix} = (-1)^3\big(4(10) - 6(7)\big) = 2$

$c_{13} = (-1)^{(1+3)} \begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix} = (-1)^4\big(4(8) - 5(7)\big) = -3$

$c_{21} =$

$c_{22} =$

$c_{23} =$

$c_{31} =$

$c_{32} =$

$c_{33} =$

$$\det(A) = \sum_{j=1}^{n} c_{ij}a_{ij} = c_{11}a_{11} + c_{12}a_{12} + c_{13}a_{13} = (2)1 + (2)2 + (-3)3 = -3$$

$$\det(A) = \textstyle\sum_{i=1}^{n} c_{ij}a_{ij} = c_{11}a_{11} + c_{21}a_{21} + c_{31}a_{31} =$$

3.5 Determinant Properties

เมตริกซ์ A และ B เป็นเมตริกซ์มิติ $n \times n$

$$\det(AB) = \det(A)\det(B)$$

$$det(A^t) = \det(A)$$

$$det(A^m) = (det(A))^m \text{ โดยที่ } m \in \mathbb{Z}^+$$

$$det(A^{-1}) = \frac{1}{\det(A)} \text{ โดยที่ } A \text{ เป็น non} - \text{singular matrix}$$

$$\det(kA) = k^n \det(A) \text{ โดยที่ } k \in \mathbb{R}$$

3.6 เมทริกซ์ผกผัน (Inverse Matrix)

นิยามที่ 3.18 เมทริกซ์ผกผัน (Inverse Matrix) คือเมทริกซ์จตุรัส ที่มีตัวผกผัน (Invertible) A^{-1} ขนาด $n \times n$ ที่ทำให้

$$AA^{-1} = I_n = A^{-1}A$$

ดังนั้นจะเรียกได้ว่า A เป็นเมทริกซ์ไม่เอกฐาน (Nonsingular Matrix) และ

ถ้า A ไม่มีตัวผกผันจะเรียกว่าเป็นเมทริกซ์เอกฐาน (Singular Matrix)

กำหนดให้ $a \in R$ ถ้า $A = [a]$ เป็นเมทริกซ์มิติ 1×1 แล้ว

$$A^{-1} = \left[\frac{1}{a}\right]$$

กำหนดให้ $a, b, c, d \in R$ เป็นจำนวนจริง ถ้า $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ เป็นเมทริกซ์มิติ 2×2 แล้ว

$$A^{-1} = \frac{1}{ad - bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix} = \frac{1}{\det(A)}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

ตัวอย่างที่ 3.18: $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ และ $B = \begin{bmatrix} -2 & 1 \\ 3/2 & -1/2 \end{bmatrix}$ จงหา AB และ BA และ A, B มีความสัมพันธ์อย่างไร

<u>วิธีทำ</u> $AB =$

$BA =$

สำหรับเมทริกซ์ A และ B มีขนาด $n \times n$ ที่หา inverse การคูณได้ ดังนั้น

$$\left(A^{-1}\right)^{-1} = A$$

$$(AB)^{-1} = B^{-1}A^{-1}$$

$$(A^{-1})^{t} = (A^{t})^{-1}$$

$$(A^{m})^{-1} = \left(A^{-1}\right)^{m}, \quad \text{โดยที่ } m \in \mathbb{Z}^{+}$$

$$(kA)^{-1} = \frac{1}{k}A^{-1}, \quad \text{โดยที่ } k \in \mathbb{R} \text{ และ } k \neq 0$$

ตัวอย่างที่ 3.19: Inverse ของการคูณ

1. ถ้า $A = [3]$ แล้ว

$\therefore \; A^{-1} = \left[\frac{1}{a}\right] = \left[\frac{1}{3}\right]$

2. ถ้า $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ แล้ว

$$\therefore A^{-1} = \frac{1}{ad - bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix} = \frac{1}{(1)4 - (2)3}\begin{bmatrix} 4 & -2 \\ -3 & 1 \end{bmatrix}$$

$$= -\frac{1}{2}\begin{bmatrix} 4 & -2 \\ -3 & 1 \end{bmatrix} =$$

นิยามที่ 3.19 เมทริกซ์ผูกพัน (Adjoint matrix) เป็นการหา Inverse ของเมทริกซ์มิติ $n \times n$ ที่กำหนดให้ A เป็นเมทริกซ์ $n \times n$ ถ้า c_{ij} เป็น co-factor ในแถวที่ i และหลักที่ j ของเมทริกซ์ A โดยที่ $i, j = 1, 2, 3, \dots, n$ แล้ว adjoint matrix ของ A เขียนแทนด้วย

$$Adj(A) = [c_{ij}]_{n \times n}^{t}$$

ทฤษฎีบทที่ 3.1 กำหนด A เป็นเมตริกซ์ $n \times n$ จะได้ว่า

$$A^{-1} = \frac{1}{\det(A)} Adj(A)$$

ตัวอย่างที่ 3.20 จากตัวอย่างที่ 3.17 ถ้าให้ $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix}$ จงหา A^{-1} จะได้ $\det(A) = -3$,

$$Adj(A) = [c_{ij}]_{n \times n}^{t} = \begin{bmatrix} 2 & 4 & -3 \\ 2 & -11 & 6 \\ -3 & 6 & -3 \end{bmatrix}$$

$$\therefore A^{-1} = \frac{1}{\det(A)} Adj(A) = -\frac{1}{3}\begin{bmatrix} 2 & 4 & -3 \\ 2 & -11 & 6 \\ -3 & 6 & -3 \end{bmatrix} = \begin{bmatrix} -\frac{2}{3} & -\frac{4}{3} & 1 \\ -\frac{2}{3} & \frac{11}{3} & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

3.7 Matrix and Python

คำสั่งต่างๆ ที่เกี่ยวข้องกับเมทริกซ์ ดังนี้

```
import numpy as np
```

1. การสร้างอาร์เรย์

```
np.array(list,type)
np.arange(from,to,increment)
np.zeros((dim1,dim2),type)
np.ones((dim1,dim2),type)
```

```
>>> import numpy as np
>>> a = np.array([[2.0, -1.0],[-1.0, 3.0]])
>>> print(a)
[[ 2. -1.]
[-1. 3.]]
>>> b = np.array([[2, -1],[-1, 3]],np.float)
>>> print(b)
[[ 2. -1.]
[-1. 3.]]
>>> print(np.arange(2,10,2))
[2 4 6 8]
>>> print(np.arange(2.0,10.0,2.0))
[ 2. 4. 6. 8.]
>>> print(np.zeros(3))
[ 0. 0. 0.]
>>> print(np.zeros((3),np.int))
[0 0 0]
>>> print(np.ones((2,2)))
[[ 1. 1.]
[ 1. 1.]]
```

2. การเข้าถึงอาร์เรย์และการเปลี่ยนแปลงค่าสมาชิกในอาร์เรย์

```
>>> a = np.zeros((3,3),np.int)
>>> print(a)
[[0 0 0]
[0 0 0]
[0 0 0]]
>>> a[0] = [2,3,2]      # เปลี่ยนแปลงค่าในแถวที่ 0
>>> a[1,1] = 5          # เปลี่ยนแปลงค่าในแถวที่ 1 หลักที่ 1
>>> a[2,0:2] = [8,-3]   # เปลี่ยนแปลงค่าบางส่วนในแถวที่ 2
>>> print(a)
```

```
[[ 2 3 2]
[ 0 5 0]
[ 8 -3 0]]
```

3. การดำเนินการกับอาร์เรย์

```
np.sqrt(array)
np.sin(array)
np.cos(array)
np.tan(array)
```

```
>>> a = np.array([0.0, 4.0, 9.0, 16.0])
>>> print(a/16.0)
[ 0. 0.25 0.5625 1. ]
>>> print(a - 4.0)
[ -4. 0. 5. 12.]
>>> a = np.array([1.0, 4.0, 9.0, 16.0])
>>> print(np.sqrt(a))
[ 1. 2. 3. 4.]
>>> print(np.sin(a))
[ 0.84147098 -0.7568025 0.41211849 -0.28790332]
```

สำหรับกรณีที่เรียกใช้คำสั่ง เช่น sqrt จากโมดูล math จะไม่สามารถดำเนินการกับอาร์เรย์ได้ ดังตัวอย่าง

```
>>> import math
>>> a = np.array([1.0, 4.0, 9.0, 16.0])
>>> print(math.sqrt(a[1]))
2.0
>>> print(math.sqrt(a))
Traceback (most recent call last):
...
TypeError: only length-1 arrays can be converted to Python scalars
```

4. Array Functions

- การหาค่าอาร์เรย์ในแนวทแยง

np.diagonal(array, k=0)

k เป็น integer ถ้า k=0 ตำแหน่งแนวทแยง ที่ i=j สำหรับเมทริกซ์จตุรัส

k<0 คือตำแหน่งแนวทแยงที่ i>j สำหรับเมทริกซ์จตุรัสและ k>0 คือตำแหน่งแนวทแยงที่ i<j สำหรับเมทริกซ์จตุรัส

- หาผลรวมในแนวทแยง

np.trace(array, k=0)

- หาตำแหน่ง index ของค่าที่มากที่สุดใน array

np.argmax(array, axis=None)

axis เป็นมิติของ array

- หาตำแหน่ง index ของค่าที่น้อยที่สุดใน array

np.argmin(array, axis=None)

- หาเมทริกซ์เอกลักษณ์ (Identity matrix)

np.identity(n, dtype=None)

n เป็นขนาดของเมทริกซ์จตุรัส

dtype เป็นชนิดของข้อมูล

ตัวอย่างการใช้คำสั่งต่างๆ ของ array function

```
>>> A = np.array([[4,-2,1],[-2,4,-2],[1,-2,3]],np.float)
>>> print(A)
[ 4. 4. 3.]
>>> b = np.array([1,4,3],np.float)
>>> print(np.diagonal(A))          # Principal diagonal
[ 4. 4. 3.]
>>> print(np.diagonal(A,1))        # First subdiagonal
[-2. -2.]
>>> print(np.trace(A))             # Sum of diagonal elements
11.0
>>> print(np.argmax(b))            # Index of largest element
1
>>> print(np.argmin(A,axis=0))     # Indices of smallest col. elements
[1 0 1]
```

```
>>> print(np.identity(3))          # Identity matrix
[[ 1. 0. 0.]
[ 0. 1. 0.]
[ 0. 0. 1.]]
```

- การหา dot product คือผลคูณของจุด

```
np.dot(A, B)
```

- การหา inner product คือผลคูณ ซึ่งหากเป็นเวกเตอร์จะได้ผลเช่นเดียวกับ dot product หากเป็นเมทริกซ์ จะเป็นการคูณของ AB^t

```
np.inner(A, B)
```

- การหา outer product
-

```
x = np.array([7,3])
y = np.array([2,1])
A = np.array([[1,2],[3,2]])
B = np.array([[1,1],[2,2]])
# Dot product
print("dot(x,y) =\n", np.dot(x,y))           # {x}.{y}
print("dot(A,x) =\n", np.dot(A,x))           # [A]{x}
print("dot(A,B) =\n", np.dot(A,B))           # [A][B]
# Inner product
print("inner(x,y) =\n", np.inner(x,y))       # {x}.{y}
print("inner(A,x) =\n", np.inner(A,x))       # [A]{x}
print("inner(A,B) =\n", np.inner(A,B))       # [A][B_transpose]
# Outer product
print("outer(x,y) =\n", np.outer(x,y))
print("outer(A,x) =\n", np.outer(A,x))
print("outer(A,B) =\n", np.outer(A,B))
```

ผลลัพธ์ของโปรแกรม

```
dot(x,y) =
17
dot(A,x) =
[13 27]
dot(A,B) =
[[5 5]
[7 7]]
inner(x,y) =
17
inner(A,x) =
[13 27]
inner(A,B) =
[[ 3 6]
[ 5 10]]
outer(x,y) =
[[14 7]
[ 6 3]]
outer(A,x) =
[[ 7 3]
[14 6]
[21 9]
[14 6]]
Outer(A,B) =
[[1 1 2 2]
[2 2 4 4]
[3 3 6 6]
[2 2 4 4]]
```

5. คำสั่งอื่นๆ

- การ copy array

```
new_array=np.copy(array)
```

คำสั่งนี้จะเป็นการสร้าง object ของ array ขึ้นมาใหม่ ซึ่งแตกต่างการใช้เครื่องหมาย assignment เช่น a=b จะไม่ได้เป็นการสร้าง object ขึ้นมาใหม่

```
>>> a = np.array([1, 2, 3])
>>> b = a
>>> c = np.copy(a)
>>> a[0] = 5
>>> a
```

```
array([5, 2, 3])
>>> b
array([5, 2, 3])
>>> c
array([1, 2, 3])
```