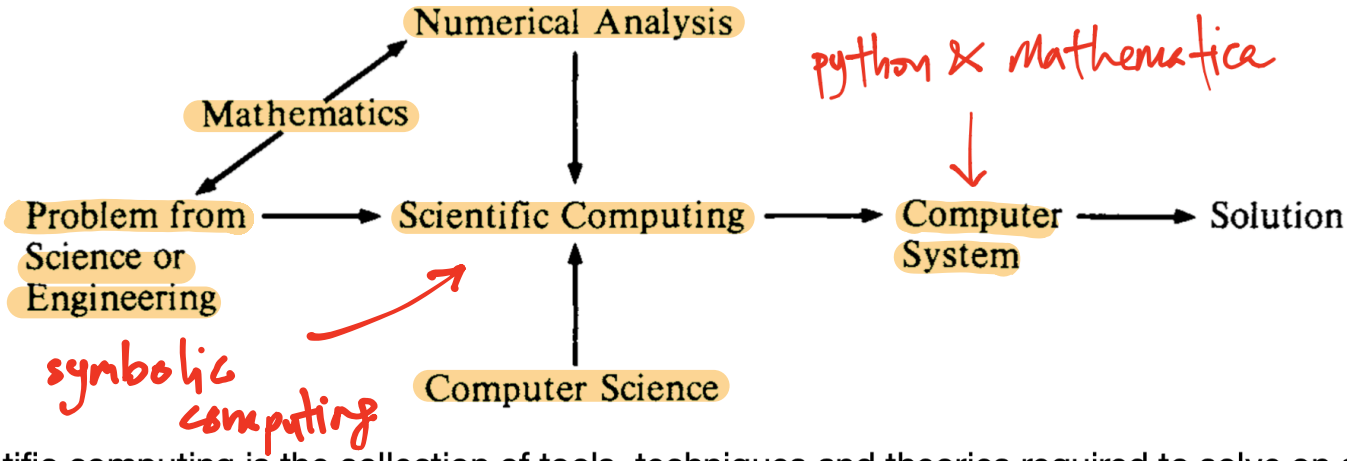
Review: Scientific Computation (MKP3303)

Gobithaasan : gr@umt.edu.my

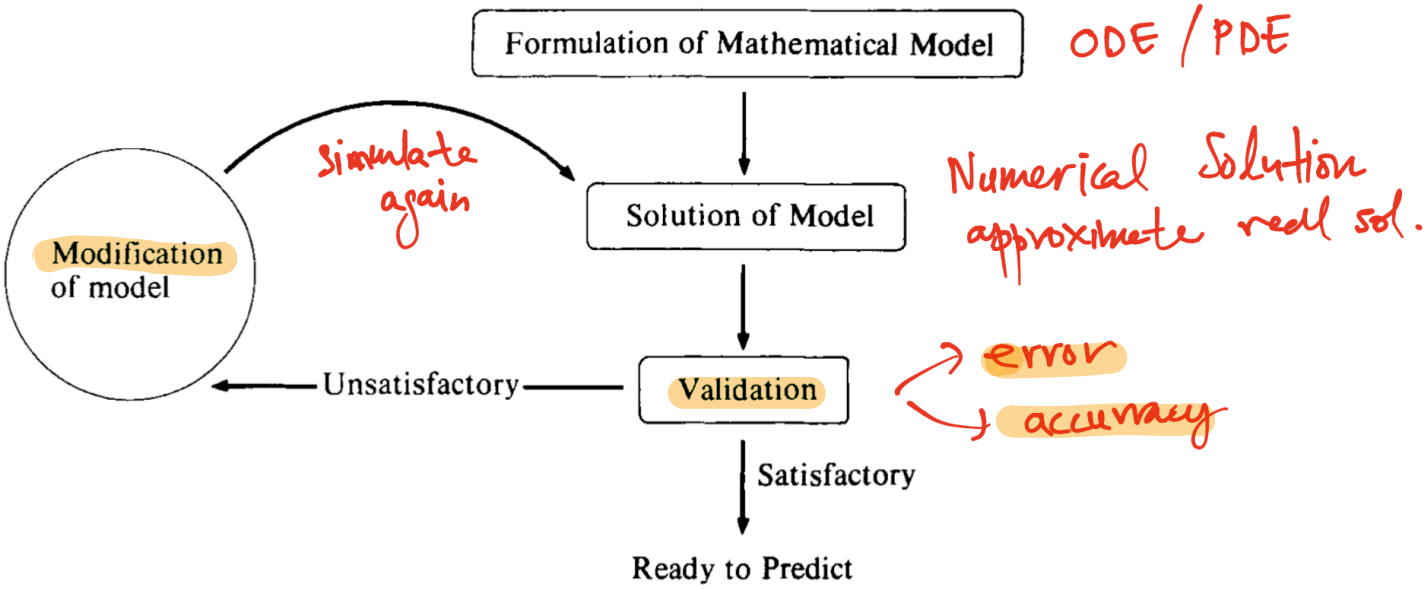
Definition of scientific computing



"Scientific computing is the collection of tools, techniques and theories required to solve on a computer the mathematical models of problems in science and engineering."

– Gene H. Golub and James M. Ortega, Scientific Computing and Differential Equations. An Introduction to Numerical Methods (1991, Elsevier Inc).

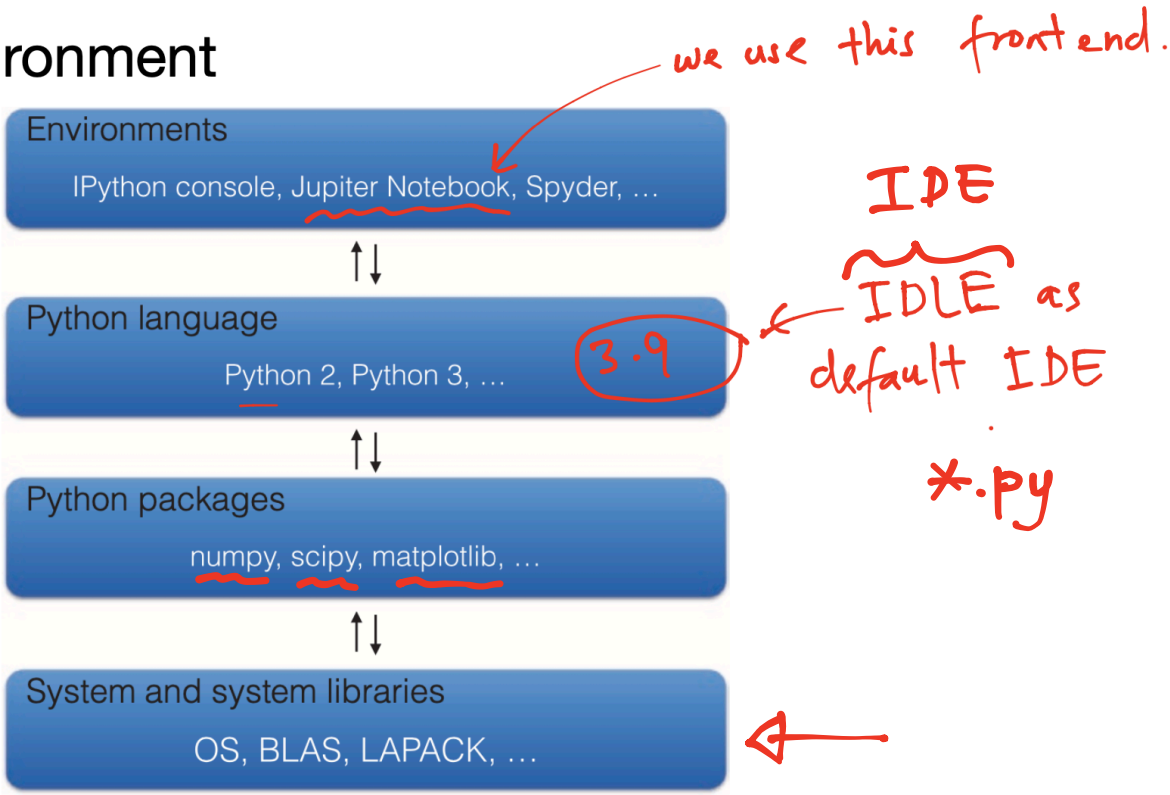
Mathematical Modeling and Solution Process



Chapter 0: Introduction to SC

- Python
- Jupyter-lab / Jupyter Notebook
- Wolfram Mathematica

Python environment



Chapter 1: Preliminaries

- Command line reference windows shell

  IDE : notebook: *.ipynb / IDLE: *.py

- Using Notebook
  - ↝ Markdown
  - ↝ code
  - ↝ raw

- Markdown Cells * for documentation
  - ↝ text
  - ↝ latex
  - ↝ html.

- Scientific Python Ecosystem *intro to modules
  - ↝ Numpy
  - ↝ Matplotlib } SciPy
  - ↝ Pandas

- Documentation and Help files

  * accessing built-in docs.

  * website PyPi : Python Package Index

- Input / Output caching

  * accessing previous input/output

  In[27], Out[61].

Chapter 2: Numbers, Expressions and Functions

- Integers, rational numbers and irrational numbers.
- Floating point numbers
- Complex numbers

type(_)

- Expressions, assignment statements & equalities.

a*b a = 2 LHS == RHS, >=

- Control Flow Statements

in, if-elif-else, while, for

- Functions (Python & Mathematics)
  - return value
  - return no value
  - built-in
  - user defined.
  - lambda expression
- Overflow error, underflow error and rounding-off error.

* floating point aritmetic

- Developing your own module

*.py

Chapter 3: Lists, Arrays, Vectors and Matrix Operations

PART 1:

- Types of Sequences in Python: Built-in containers

1. str : " " or ' '
2. list : [ ]
3. tuple : ( ) $(x,y) \neq (y,x)$ $\in 2D$.
4. set : { } $\{x,y\} = \{y,x\}$
5. dict : { key1 : val1, _ : _ }
   dictionary

PART 2:

vectorized computing

- Arrays NumPy : import numpy as np
- Column and row vector [ ] .
- Matrix representation [[ ],[ ]].

subpackages: linalg

→ np.array([_])

→ _.shape

→ _.reshape

matrix operation :

(a) +/− : $m_1 \pm m_2$

(b) element wise : 5 * m

(c) dot product : np.dot(m_1, m_2) : m_1 @ m_2

(d) cross product : np.cross(m_1, m_2)

PART 3:

- Introduction to array operations
- Vector and Matrix Operations
- Towards Higher dimensions
- Reading and writing files
- Bonus: Python for Data Analysis (Pandas)

→ csv (comma) / txt / tsv (tab)

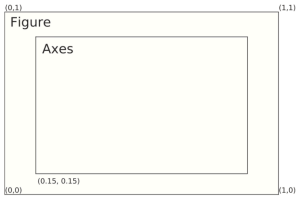
→ Dataframe

Chapter 4: Data Visualization

NumPy

Static Visualization: (MatPlotLib, Pandas, Seaborn, plotnine)

matplotlib



- 2D: Two-dimensional plot (Plane)
- 3D: Three-dimensional plot (Space) { subplot_kw={'projection':'3d'} add_subplot(projection='3d').
- Time Series Visualization

↳ Import pandas as pd. ↳ pd.Series.

↳ Import seaborn as sns.

Interactive Visualization: (Bokeh, Plotly) ↗ plotly.express ↘ plotly.graph_objects

↖ Java Script Lib.

- 2D: Two-dimensional plot (Plane)
- 3D: Three-dimensional plot (Space)
- Higher dimension Visualization ⇝ annotations:

x_2 ↑ → x_1 (x_1, x_2, x_3, x_4)

① size ② shape ③ color.

Dimension Reduction with PCA

- 2D: Two-dimensional plot (Plane)

↳ option ①: use scikit-learn library.

↳ option ②: manual: ① standardization: $\tilde{x} = x_i \in X - \text{mean}(X)$

② covariance cv = np.cov($\tilde{X}$)

③ eigen vectors / values: np.linalg.eig(cv)

④ Project $\tilde{X}$ to 2D using eigen vectors / values.

np.dot($\tilde{X}$, eig_vec[:,0])

Additional: Graph Theory and Network Visualization

Chapter 5: Integration

import sympy

x = sympy.Symbol('x')

f = sympy.Function('f')(x)

Symbolic Computing

- Symbolic computing: SymPy

① substitution ② expand, factor, collect ④ simplify

f.subs(x, 2) ③ apart, together, cancel ⑤ plotting

Analytical Solutions with SymPy

- Indefinite integral $\int f(x)\,dx$: sympy.integrate(f, x)
- Definite integral $\int_a^b f(x)\,dx$: sympy.integrate(f, (x, a, b))

sympy.oo : ∞

Numerical Solutions with SciPy

from scipy import integrate

- Definite integral
- Integration and its application

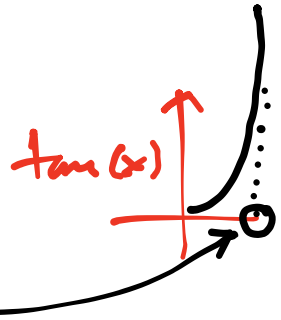
* f = lambda x : np.f(x)

val, err = integrate.quad(f, a, b)

Gaussian Quadrature

Improper Integrals : having singularities.

* integrate.quad(f, a, b, points = [_])



Multiple Integrals: → integrate.dblquad.

→ integrate.tplquad.

Chapter 6: Solution of Equation: Differentiation

- Limits

sympy.limit(f, x, ___)

x → ∞ (sympy.oo)

x → limit

- Differentiation: derivatives and ODEs.

⊛ sympy.diff(f, x, ___) number of times of diff

⊛ ODE : f(x, y, y') = y' − 2x

```
ode = y(x).diff(x) - 2*x
sympy.dsolve(ode).
```

⊛ direction field

- Analytical Solutions of equation(s)

⊛ one variable : sympy.solve(f(x), x)

⊛ linear system : np.linalg.lu(A, b)

sympy.solve_linear_system(A|b, x1, ...)

- Numerical Solutions of equation(s)

⊛ one variable :

sympy.solve / optimize.bisect(f, a, b) (optimize: Scipy)

scipy.optimize.newton(f, x*, fp)

if empty, then its secant method.

⊛ nonlinear system : sympy.solve([_], x1, ...)

Powell hybrid

fsolve(func, [_, _, ...])

initial values.

system of nonlinear equation : def func:
return [eqn1,
eqn2,
:
eqnn]

with fprime :

fsolve(func, [_, _ ..], fprime = ___)

Jacobian matrix.

derive using sympy's jacobian(_)

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & & \\ \frac{\partial f_m}{\partial x_1} & \cdots & & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$