Mục lục

1 Chuẩn bị **1**

1.1 Kiến thức về giải tích 1
1.2 Sai số làm tròn và số học máy tính 3
1.3 Thuật toán và sự hội tụ 3
1.4 Python: ngôn ngữ tính toán và lập trình 3
1.5 Python + VS Code: giải tích và đại số 11

2 Giải phương trình một biến **22**

2.1 Phương pháp chia đôi 22
2.2 Phương pháp Newton và mở rộng 24
2.3 Lặp điểm bất động 30
2.4 Phân tích sai số của các phương pháp lặp 35
2.5 Tăng tốc độ hội tụ 35
2.6 Nghiệm của đa thức và phương pháp Müller 35

3 Nội suy và xấp xỉ bằng đa thức **30**

3.1 Nội suy tổng quát 30
3.2 Đa thức nội suy 31
3.3 Xấp xỉ số liệu và phương pháp Neville 35
3.4 Sai phân chia 35
3.5 Nội suy Hermite 35
3.6 Nội suy Newton 35
3.7 Nội suy spline bậc ba 39
3.8 Đường cong tham số 39

4 Đạo hàm và tích phân bằng số **40**

4.1 Đạo hàm bằng số 41
4.2 Ngoại suy Richardson 45
4.3 Tích phân bằng số 45
4.4 Tích phân Romberg 50

4.5 Phương pháp cầu phương thích ứng . . . 50
4.6 Cầu phương Gauss . . . 50
4.7 Tích phân bội . . . 50
4.8 Tích phân suy rộng . . . 50

5 Bài toán giá trị ban đầu của phương trình vi phân thường 51
5.1 Lý thuyết cơ bản về bài toán giá trị ban đầu . . . 52
5.2 Phương pháp Picard . . . 53
5.3 Phương pháp chuỗi Taylor . . . 57
5.4 Phương pháp Euler . . . 59
5.5 Phương pháp Taylor bậc cao . . . 62
5.6 Phương pháp Runge–Kutta . . . 63
5.7 Điều khiển sai số và phương pháp Runge–Kutta–Fehlberg . . . 67
5.8 Phương pháp đa bước . . . 67
5.9 Phương pháp đa bước với bước nhảy biến thiên . . . 67
5.10 Phương pháp ngoại suy . . . 67
5.11 Phương trình cấp cao và hệ phương trình vi phân . . . 67
5.12 Sự ổn định . . . 67
5.13 Phương trình vi phân cứng . . . 67

6 Phương pháp trực tiếp giải hệ phương trình tuyến tính 68
6.1 Hệ phương trình tuyến tính . . . 68
6.2 Chiến thuật chốt . . . 69
6.3 Đại số tuyến tính và ma trận nghịch đảo . . . 69
6.4 Định thức của ma trận . . . 69
6.5 Phân tích ma trận . . . 69
6.6 Các dạng ma trận đặc biệt . . . 69

7 Kỹ thuật lặp trong đại số tuyến tính 70
7.1 Chuẩn của véctơ và ma trận . . . 70
7.2 Giá trị riêng và véctơ riêng . . . 72
7.3 Lặp điểm bất động . . . 72
7.4 Kỹ thuật lặp Jacobi và Gauss–Seidel . . . 76
7.5 Ma trận nghịch đảo . . . 79
7.6 Kỹ thuật giảm dư giải hệ tuyến tính . . . 80
7.7 Giới hạn sai số và tinh chỉnh phép lặp . . . 80
7.8 Phương pháp gradient liên hợp . . . 80

8 Lý thuyết xấp xỉ **81**
8.1 Xấp xỉ bình phương nhỏ nhất . . . 81
8.2 Đa thức trực giao và xấp xỉ bình phương nhỏ nhất . . . 85
8.3 Đa thức Chebyshev và [Economization] chuỗi lũy thừa . . . 86
8.4 Xấp xỉ hàm hữu tỷ . . . 86
8.5 Xấp xỉ đa thức lượng giác . . . 86
8.6 Biến đổi Fourier nhanh . . . 86

9 Xấp xỉ giá trị riêng **84**
9.1 Đại số tuyến tính và giá trị riêng . . . 84
9.2 Ma trận trực giao và biến đổi đồng dạng . . . 84
9.3 Phương pháp lũy thừa . . . 84
9.4 Phương pháp Householder . . . 84
9.5 Thuật toán QR . . . 84
9.6 Phân tích giá trị kỳ dị . . . 84

10 Nghiệm số của hệ phương trình phi tuyến **85**
10.1 Điểm bất động của hàm nhiều biến . . . 85
10.2 Phương pháp Newton . . . 85
10.3 Phương pháp tựa Newton . . . 85
10.4 Phương pháp độ dốc nhất . . . 85
10.5 Đồng luân và các phương pháp mở rộng . . . 85

11 Bài toán giá trị biên của phương trình vi phân thường **86**
11.1 Phương pháp bắn tuyến tính . . . 86
11.2 Phương pháp bắn cho bài toán phi tuyến . . . 86
11.3 Phương pháp sai phân hữu hạn cho bài toán tuyến tính . . . 86
11.4 Phương pháp sai phân hữu hạn cho bài toán phi tuyến . . . 87
11.5 Phương pháp Rayleigh–Ritz . . . 87

12 Nghiệm số của phương trình đạo hàm riêng **88**
12.1 Phương trình đạo hàm riêng Elliptic . . . 88
12.2 Phương trình đạo hàm riêng Parabolic . . . 89
12.3 Phương trình đạo hàm riêng Hyperbolic . . . 89
12.4 Giới thiệu về phương pháp phần tử hữu hạn . . . 89

Chương 1

Chuẩn bị

1.1 Kiến thức về giải tích 1
1.2 Sai số làm tròn và số học máy tính 3
1.3 Thuật toán và sự hội tụ 3
1.4 Python: ngôn ngữ tính toán và lập trình 3
1.5 Python + VS Code: giải tích và đại số 11

1.1 Kiến thức về giải tích

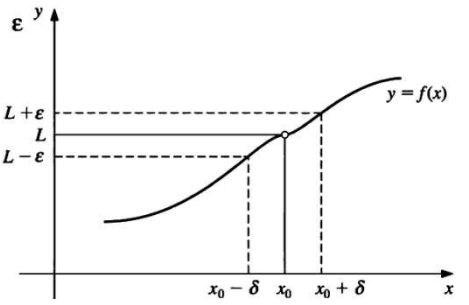
1.1.1 Giới hạn và tính liên tục

Định nghĩa 1.1. *Hàm f xác định trên tập X các số thực cố **giới hạn** L tại x_0, ký hiệu*

$$\lim_{x \to x_0} f(x) = L$$

nếu với mọi $\varepsilon > 0$, tồn tại $\delta > 0$ sao cho

$$|f(x) - L| < \varepsilon \quad \text{với mọi} \quad x \in X \quad \text{và} \quad 0 < |x - x_0| < \delta.$$



Định nghĩa 1.2. *Cho hàm f xác định trên tập X các số thực và $x_0 \in X$. f* ***liên tục*** *tại x_0 nếu*

$$\lim_{x \to x_0} f(x) = f(x_0).$$

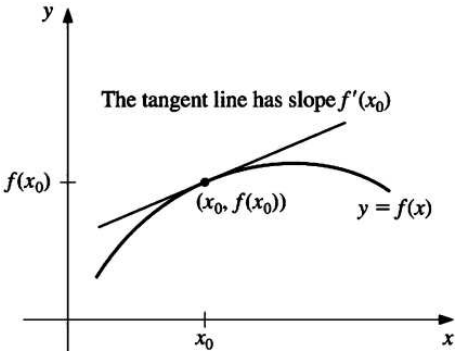
f ***liên tục trên X*** *nếu nó liên tục tại mọi điểm của X.*

1.1.2 Tính khả vi

Định nghĩa 1.3. *Cho hàm f xác định trên khoảng mở chứa x_0. f* ***khả vi*** *tại x_0 nếu tồn tại*

$$f'(x_0) = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0}.$$

$f'(x_0)$ gọi là ***đạo hàm*** *của f tại x_0. Hàm có đạo hàm tại mọi điểm của X gọi là* ***khả vi trên X****.*



Định lý 1.1. *Giả sử $f \in C^n[a,b]$, tồn tại $f^{(n+1)}$ trên $[a,b]$, và $x_0 \in [a,b]$. Khi đó, với mọi $x \in [a,b]$, tồn tại số $\xi(x)$ ở giữa x_0 và x sao cho*

$$f(x) = P_n(x) + R_n(x)$$

trong đó

$$P_n(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$
$$= \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k$$

và

$$R_n(x) = \frac{f^{(n+1)}\left(\xi(x)\right)}{(n+1)!}(x - x_0)^{n+1}.$$

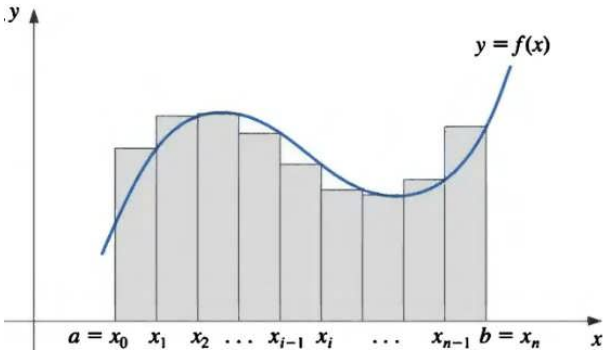
$P_n(x)$ là **đa thức Taylor bậc n** của f quanh x_0, và $R_n(x)$ là **phần dư** (hay **sai số cụt**) của $P_n(x)$.

1.1.3 Tích phân

Định nghĩa 1.4. ***Tích phân Riemann*** *của hàm f trên đoạn $[a, b]$ là giới hạn sau, miễn là nó tồn tại*

$$\int_a^b f(x)\,dx = \lim_{\max \Delta x_i \to 0} \sum_{i=1}^{n} f(z_i)\,\Delta x_i$$

trong đó $a = x_0 \leq x_1 \leq \ldots \leq x_n = b$, $\Delta x_i = x_i - x_{i-1}$ với $i = 1, 2, \ldots, n$, và $z_i \in [x_{i-1}, x_i]$ tùy ý.



1.2 Sai số làm tròn và số học máy tính

1.2.1 Số nhị phân máy

1.2.2 Số thập phân máy

1.2.3 Số học hữu hạn chữ số

1.2.4 Số học lồng nhau

1.3 Thuật toán và sự hội tụ

1.4 Python: ngôn ngữ tính toán và lập trình

1.4.1 Lý do chọn Python

Các ngôn ngữ tính toán và lập trình mạnh và phổ biến như MATLAB, Mathematica, Python, Maple, tích hợp các lệnh, gói lệnh giải các bài toán phổ biến. Nắm vững thuật toán giúp ta triển khai giải bài toán trên các ngôn ngữ khác như C, C++, Java, FORTRAN,...

Thông tin	MATLAB	Mathematica	Python
Năm ra đời	1989	1986	1989
Tác giả, công ty	MathWorks	Wolfram Research	Guido van Rossum
Hệ điều hành: Windows (1), macOS (2), Linux (3), Android (4), iOS (5), Raspberry Pi (6)	1, 2, 3	1, 2, 3, 6	tất cả
Phiên bản năm 2021	R2021a (9.10)	12.2	3.9.5
Giá	49 – 2 150$ không kèm Toolbox	177 – 5 780$/năm, miễn phí trên (6)	miễn phí, mã nguồn mở
Dung lượng tải – cài đặt trên Windows	20.8 – 30.5GB	4.3 – 11.9GB	27 – 100.6MB
Độ phổ biến theo chỉ số PYPL*	1.71%		29.9%
Độ phổ biến theo chỉ số TIOBE	1.23%		11.87%

Python có khả năng tính toán mạnh mẽ, ngôn ngữ dễ hiểu, dễ lập trình, nhiều môi trường phát triển tích hợp (IDE), cộng đồng sử dụng lớn,... Hầu hết các bài toán đề cập trong cuốn sách, với sự hỗ trợ của Python, đều được giải quyết ngắn gọn, mà không đòi hỏi ta phải nhớ quá nhiều kiến thức toán học.

Các bước tải và cài Python:

1. Duyệt trang https://www.python.org → trỏ chuột vào thẻ *Downloads* → nhấp vào nút *Python 3.11.2* để tải về.

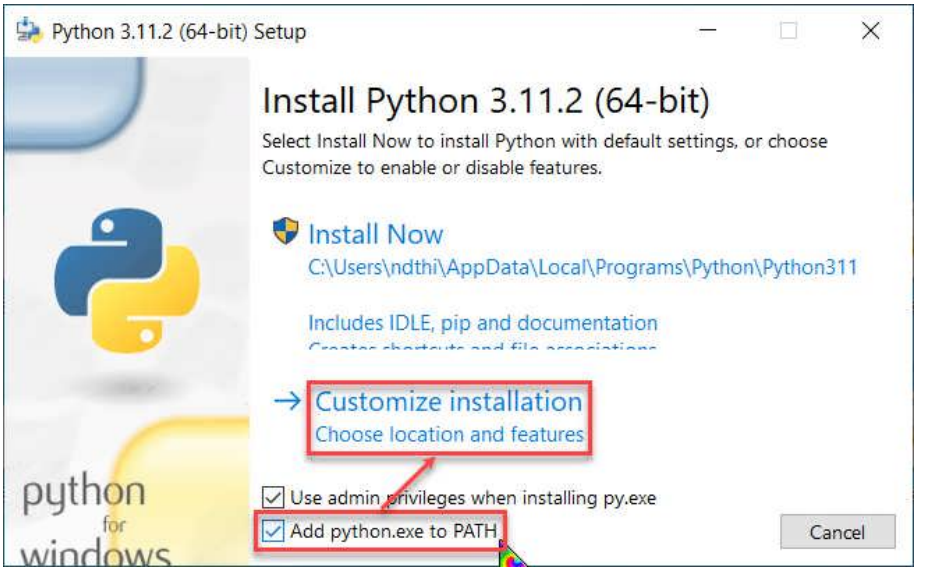
   Trình duyệt tự động nhận dạng phiên bản hệ điều hành và cho tải phiên bản Python tương ứng†.

Python	Windows
$\leq$ 3.4.x	Mọi phiên bản Windows
3.5.x → 3.8.x	> Windows XP
3.9.x, 3.10.x	> Windows 7

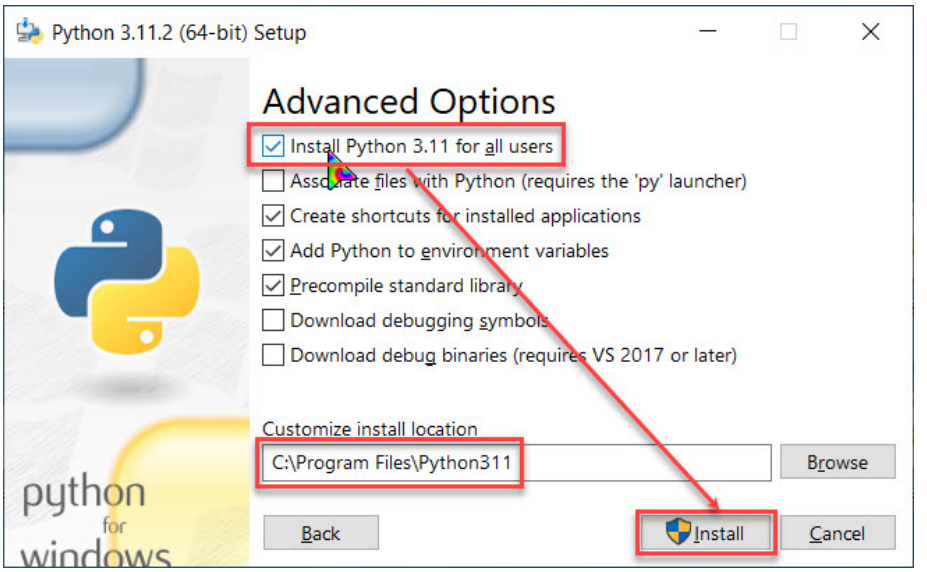
*Số liệu lấy từ https://statisticstimes.com/

†Trên Linux, Python được tích hợp sẵn

2. Mở file *python-3.11.2-amd64.exe* vừa tải về → tick chọn *Add python.exe to PATH* để khai báo môi trường cho hai tệp thực thi quan trọng là *python.exe* và *pip.exe* → *Customize installation*. Ở hộp thoại *Optional Features* nhấn *Next*.



3. Hộp thoại *Advanced Options* tick chọn *Install Python 3.11 for all users* → nhớ đường dẫn cài đặt Python → *Install*.

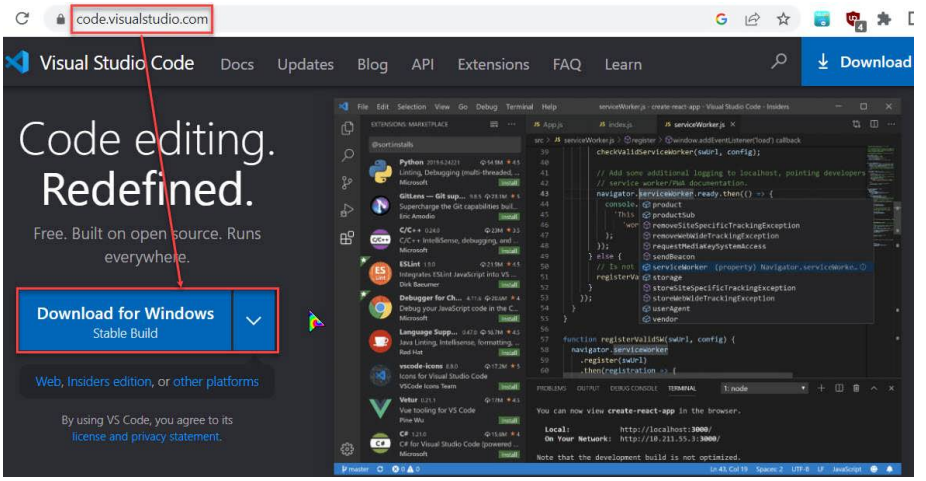


1.4.2 Visual Studio Code (VS Code)

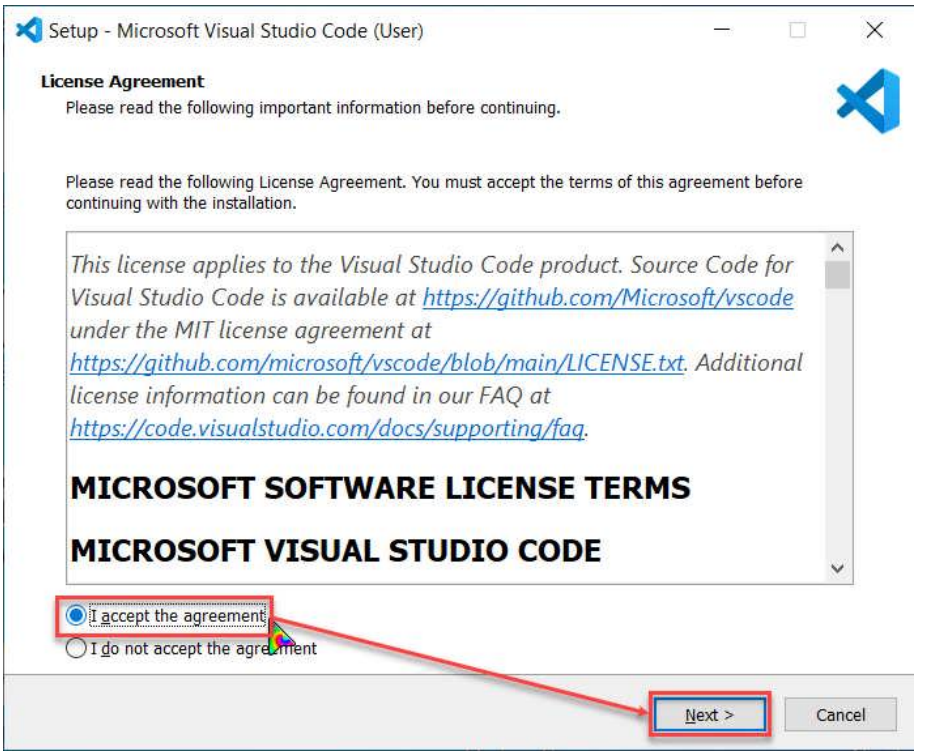
Đây là một môi trường phát triển tích hợp (IDE) mạnh mẽ, hỗ trợ nhiều ngôn ngữ lập trình.

Các bước tải, cài đặt và sử dụng VS Code:

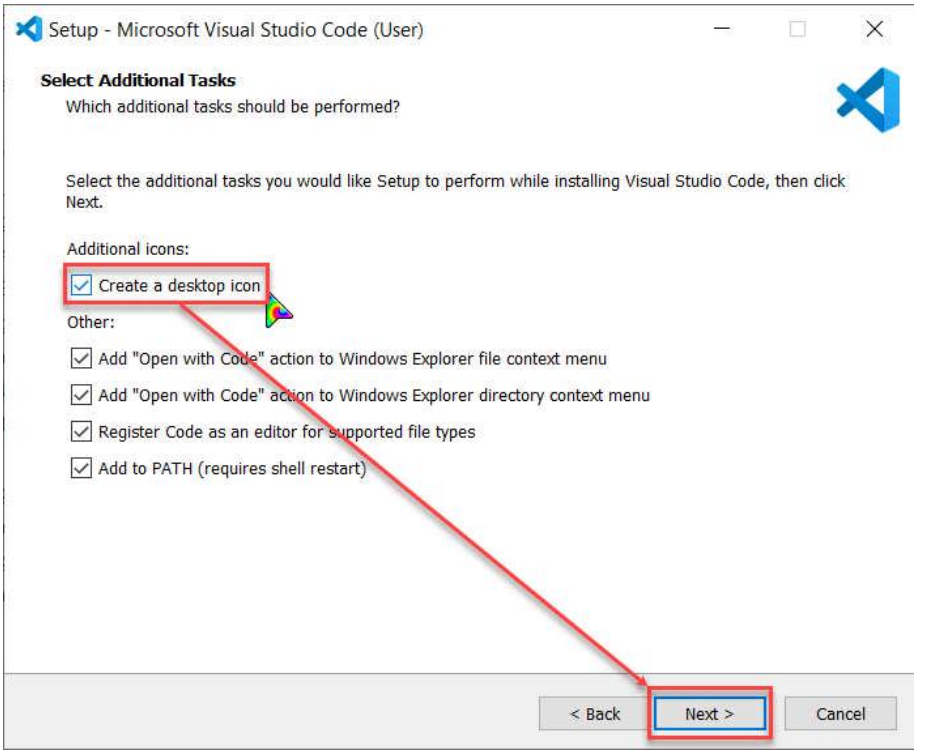
1. Duyệt trang https://code.visualstudio.com → *Download for Windows*.



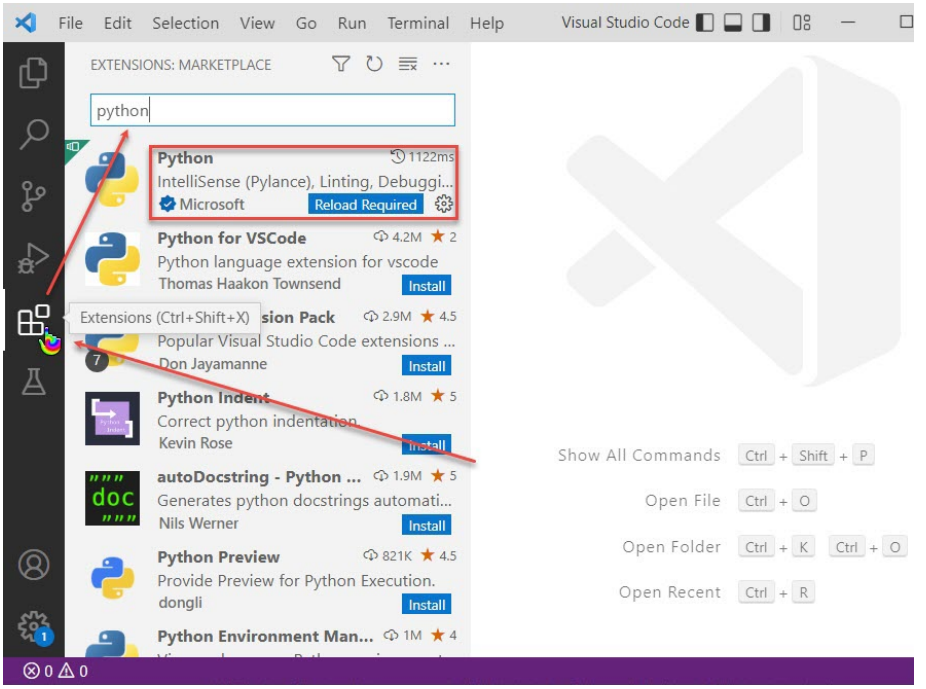
2. Mở tệp *VSCodeUserSetup-x64-1.76.2.exe* → chọn *I accept the agreement* → *Next*.



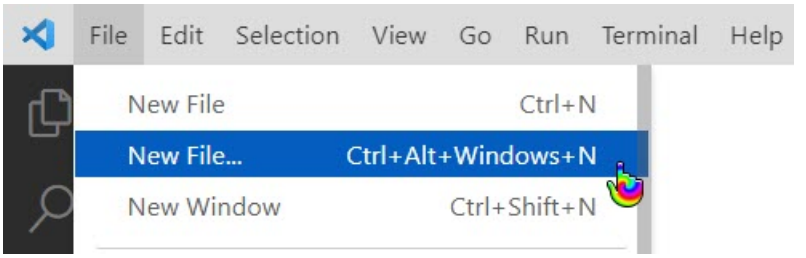
3. Hộp thoại *Select Additional Tasks* có thể chọn *Create a desktop icon* → *Next* → *Install*.



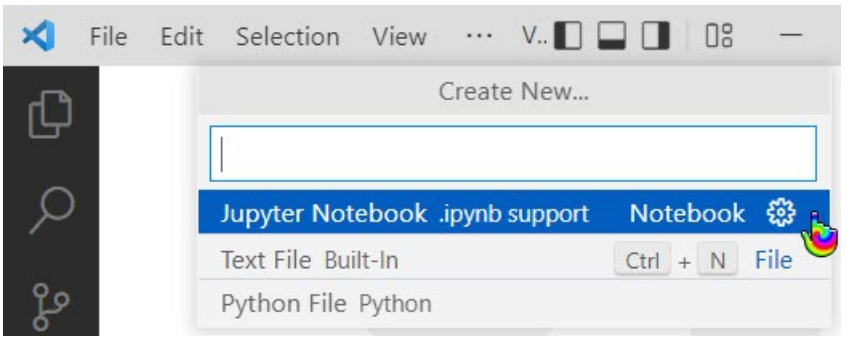
4. Cài phần mở rộng (1) *Python* và (2) *Jupyter* để lập trình Python trên sổ tay Jupyter Notebook. Bước này cần có kết nối mạng.



5. Để tạo sổ tay mới, ta vào menu *File* → *New File*... (mục thứ hai)

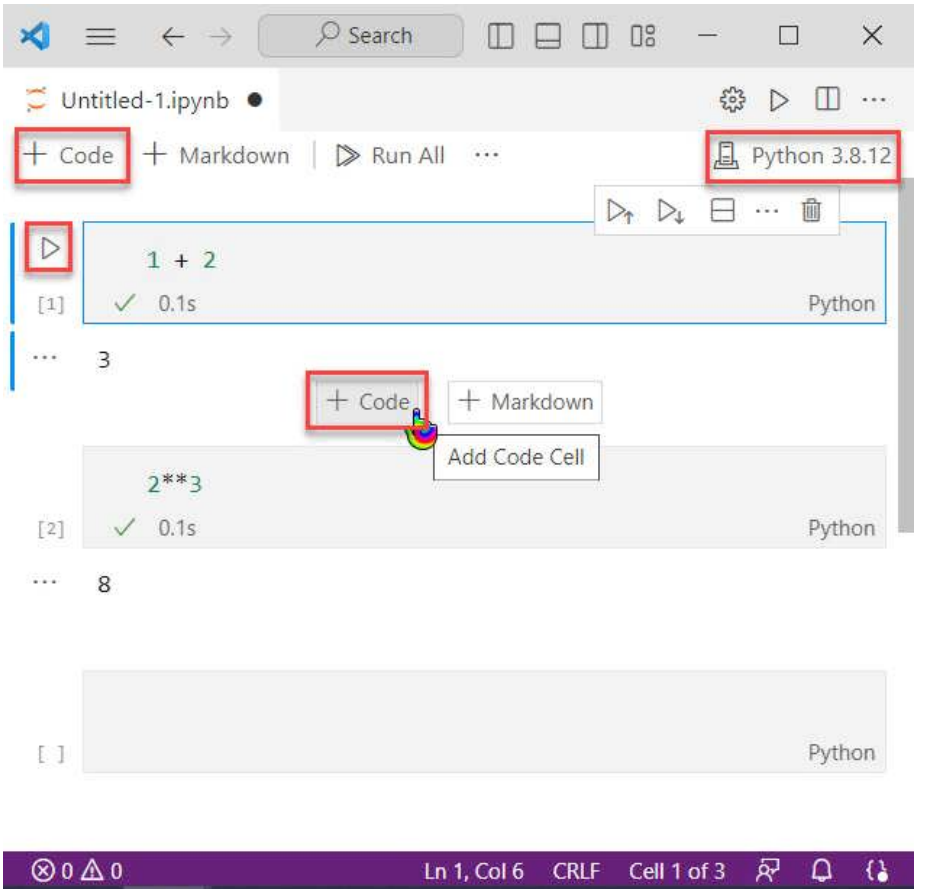


và chọn loại file muốn tạo



6. Mỗi sổ tay gồm các ô lệnh, mỗi ô thường gồm hai phần: các lệnh và các kết quả. Các sổ tay là độc lập, tức là các đối tượng, dù cùng tên, trong hai sổ tay khác nhau thì cũng không liên quan gì đến nhau*.

*Trong Mathematica, các sổ tay là liên thông, tức là đối tượng nhận giá trị nào trên sổ tay này, thì cũng nhận giá trị đó trên các sổ tay khác



- Góc trên bên trái hiển thị phiên bản Python
- Để chạy lệnh trong một ô, (1) nhấn nút *Execute Cell* hình tam giác ở bên trái ô đó, hoặc (2) nhấn *Shift + Enter*, hoặc (3) *Ctrl + Alt + Enter*.

  Các ô nhớ sẽ lưu giá trị của đối tượng theo thứ tự được chạy của ô, chứ không phải theo thứ tự từ trên xuống.
- Để thêm ô mới, (1) nhấn nút *Add Code Cell* hình dấu + ở góc trên bên trái (nút này cũng xuất hiện khi trỏ chuột vào đáy mỗi ô), hoặc (2) nhấn *Shift + Enter*.

  Mỗi lệnh nên đặt trong một ô, trừ khi các lệnh tham gia vào việc thay đổi giá trị của biến.

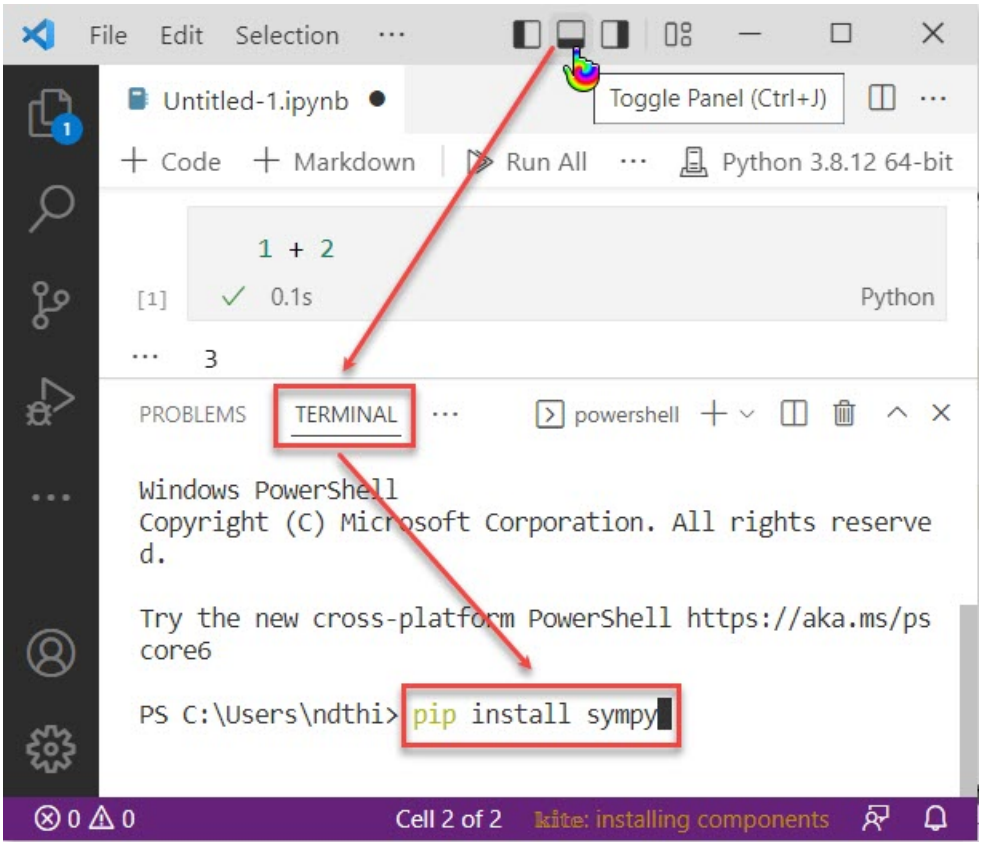
1.4.3 Gói lệnh

Một đặc điểm nổi bật của Python là kho gói lệnh phong phú, cài đặt đơn giản, và hướng dẫn sử dụng chi tiết, dễ hiểu.

Để cài thêm gói lệnh cho Python*, trong VS Code, ta nhấn vào nút *Toggle Panel* ở cụm

*Trên Google Colab, hầu hết các gói phổ biến đã được cài sẵn

nút bên phải của thanh tiêu đề, chọn thẻ *Termial* để mở cửa sổ dòng lệnh.



rồi nhập lệnh cài gói cần dùng, theo cú pháp

```
pip install gói_lệnh
```

Môn học sử dụng ba gói *sympy*, *scipy*, và *matplotlib*. Khi cài *scipy* hoặc *matplotlib*, một gói phụ thuộc quan trọng là *numpy* cũng được cài theo. Khi cần dùng lệnh của gói nào, ta khai báo gói đó, theo cú pháp

```
1 from sympy import *
2 import numpy as np
3 import matplotlib.pyplot as plt
```

- Dòng 1, dấu * ngụ ý nạp tất cả các lệnh của gói `sympy`, ngoại trừ các *module* con. Lệnh `func(...)` của gói được gọi bằng cách gõ chính tên của lệnh đó

  `func(...)`

- Dòng 2, ta nạp gói numpy dưới dạng một đối tượng và đặt tên là `np`. Mỗi lệnh của gói xem là một trường hay phương thức của gói đó. Lệnh có dạng `func(...)` được gọi bằng cú pháp

```
np.func(...)
```

- Dòng 3 tương tự dòng 2, trong đó `pyplot` là module con của gói `matplotlib`.

1.5 Python + VS Code: giải tích và đai số

Trong phần này, các cú pháp trong Python được diễn đạt tổng quát, hoặc minh họa bằng ví dụ cụ thể.

1.5.1 Phép toán số học / logic

Phép toán	Lệnh Python
$a + b$	`a + b`
$a - b$	`a - b`
ab	`a * b`
$\frac{a}{b}$	`a / b`
a^b	`a ** b`

theo thứ tự ưu tiên `**` $\to$ `*`, `/` $\to$ `+`, `-`, và $\neg, \wedge, \vee$. Ngoài ra, với biểu thức phức tạp, ta dùng dấu nhóm biểu thức `( )`.

1.5.2 Lệnh thường dùng

Lệnh	Kết quả	Mô tả
`round(7.019, 2)`	7.02	làm tròn lấy 2 chữ số sau dấu phảy
`abs(-3)`	$\lvert -3 \rvert = 3$	
`min(3, 1, -2)` `min([3, 1, -2])`	-2	số nhỏ nhất trong các số
`max(0, 4, 1)` `max([0, 4, 1])`	4	số lớn nhất của dãy
`sum([0, 4, 1])`	5	tổng của dãy

1.5.3 Dãy

Lệnh	Kết quả	Mô tả
`a = [4, 1, -2]`		khai báo dãy a gồm các phần tử $4, 1, -2$
`len(a)`	3	số phần tử, hay độ dài, cỡ của dãy a
`a[0]`	4	phần tử đầu của dãy, với chỉ số là 0

`list( range(5) )`	dãy [0, 1, 2, 3, 4]
`list( range(2, 5) )`	dãy [2, 3, 4]
`list( range(1, 10, 2) )`	dãy [1, 3, 5, 7, 9]
`[ 2*i for i in [1, 2, 3] ]`	dãy [2, 4, 6]
`[ i**2 for i in range(3) ]`	dãy [0, 1, 4]

Ghép cặp các phần tử theo vị trí tương ứng của hai dãy

```
X = [1, 2, 3]
Y = [4, 5, 6]        # cỡ của hai dãy có thể khác nhau
list( zip(X, Y) )    # [(1, 4), (2, 5), (3, 6)]
```

Nếu dãy a độ dài n, gồm các phần tử $a_0, a_1, \ldots, a_{n-1}$, thì

Lệnh	Mô tả
`a[:i]`	dãy con $a_0, a_1, \ldots, a_{i-1}$
`a[i:]`	dãy con $a_i, a_{i+1}, \ldots, a_{n-1}$
`a[i:j]`	dãy con $a_i, a_{i+1}, \ldots, a_{j-1}$

1.5.4 Biến

Cú pháp khai báo biến

```
var = expr
```

trong đó `expr` có thể là biểu thức toán học, xâu,... Chẳng hạn, lệnh

```
x = 1
x + 2
```

cho kết quả là 3.

1.5.5 Khai báo hàm

Khai báo hàm bằng từ khóa `lambda`, hoặc `def` nếu cần lập trình, hoặc đôi khi kết hợp cả hai.

Ví dụ 1.1. $f(x) = x^2$.

Cách 1:

```
f = lambda x: x**2
```

Cách 2:

```
def f(x):
    return x**2
```

Khi đó lệnh

```
f(-3)
```

cho kết quả là 9.

Ví dụ 1.2. $f(x, y) = x + y$.

Cách 1:

```
f = lambda x, y: x + y
```

Cách 2:

```
def f(x, y):
    return x + y
```

Khi đó lệnh

```
f(1, 2)
```

cho kết quả là 3.

Hai ví dụ về khai báo hàm rẽ nhánh.

Ví dụ 1.3. $|x| = \begin{cases} x & \text{nếu } x \geq 0 \\ -x & \text{nếu } x < 0 \end{cases}$.

Cách 1:

```
my_abs = lambda x: x if x >=0 else -x
```

Cách 2:

```
def my_abs(x):
    if x >= 0:
        return x
    else:
        return -x
```

Khi đó lệnh

```
my_abs(-1)
```

cho kết quả là 1.

Ví dụ 1.4. Định nghĩa hàm dấu $\operatorname{sign}(x) = \begin{cases} 1 & \text{nếu } x > 0 \\ 0 & \text{nếu } x = 0 \\ -1 & \text{nếu } x < 0 \end{cases}$.

Cách 1:

```
sign = lambda x: 1 if x > 0 else 0 if x == 0 else -1
```

Cách 2:

```
def sign(x):
    if x > 0:
        return 1
    elif x == 0:
        return 0
    else:
        return -1
```

Khi đó lệnh

```
sign(-2)
```

cho kết quả là -1.

1.5.6 Gói lệnh sympy

Gói lệnh hỗ trợ tính toán với biểu thức symbolic, tức là biểu thức chứa ký hiệu, hay biến bất định. Khai báo gói bằng lệnh

```
from sympy import *
```

trong đó dấu $*$ được hiểu bao gồm tất cả các lệnh của gói, không bao gồm các module con. Ta gọi các lệnh này bằng cách gõ trực tiếp tên lệnh đó.

Lệnh thường dùng

sympy	Kết quả	Mô tả
`E`	e	
`pi`	π	
N(pi, 6)	3.14159	làm tròn π lấy 6 chữ số có nghĩa
`factorial(3)`	3! = 6	

Biến, biểu thức symbolic

Không như biến thông thường trong các ngôn ngữ lập trình, biến symbolic không có giá trị cụ thể, ta có thể gọi là biến bất định. Biểu thức toán học chứa biến symbolic gọi là biểu thức symbolic. Để làm việc với biểu thức symbolic, trước hết cần khai báo các biến symbolic có trong biểu thức đó.

- Khai báo một hoặc vài biến cụ thể

```
x = symbols('x')
x, y = symbols('x y')
```

- Khai báo dãy các biến symbolic a_0, a_1, a_2

```
a = [ symbols(f'a{i}') for i in range(3) ]
```

Khi đó lệnh

```
a[0]
```

cho kết quả a_0.

- Khai báo một ma trận symbolic M cỡ 3×4

```
M = MatrixSymbol('M', 3, 4)
```

Khi đó lệnh

```
M[0, 2]
```

cho kết quả $M_{0,2}$ ở hàng 0 cột 2.

- Khai báo hàm bất định f

```
f = symbols('f', cls=Function)
```

Khi đó lệnh

```
f(1)
```

cho kết quả $f(1)$.

Hàm sơ cấp

Python mặc định không hỗ trợ hàm sơ cấp. Các hàm đó được tích hợp trong gói `sympy`.

Hàm	sympy
$\sqrt{x}$	`sqrt(x)`
$\sqrt[n]{x^m}$	`x ** (m/n)` hoặc `x ** Rational(m, n)`

$\sin x$, $\cos x$, $\tan x$, $\cot x$	`sin(x)`, `cos(x)`, `tan(x)`, `cot(x)`
$\arcsin x$, $\arccos x$, $\arctan x$, $\operatorname{arccot} x$	`asin(x)`, `acos(x)`, `atan(x)`, `acot(x)`
$\ln x$	`log(x)` hoặc `ln(x)`
$\log_a x$	`log(x, a)` hoặc `ln(x, a)`

Xử lý biểu thức

Python	Kết quả	Mô tả
`( sin(x) * cos(x) ).simplify()`	$\dfrac{\sin 2x}{2}$	rút gọn
`( x**2 - 3*x + 2 ).factor()`	$(x-1)(x-2)$	phân tích thành nhân tử
`( (x + 1)**2 ).expand()`	x^2+2x+1	khai triển đa thức

Tính giá trị của biểu thức

Python	Kết quả
`(x**3).subs(x, 2)`	$2^3 = 8$
`(x + y).subs({x: 1, y: -2})`	$1+(-2) = -1$

Khi biểu thức đã được khai báo bởi hàm, xem lại phần 1.5.5.

Đạo hàm

Python	Kết quả
`(x**3).diff()`	$\dfrac{\partial}{\partial x}\left(x^3\right) = 3x^2$
`(x**3).diff(x, 2)`	$\dfrac{\partial^2}{\partial x^2}\left(x^3\right) = 6x$

Tích phân

Python	Kết quả
`(x**2).integrate((x, 0, 1))`	$\displaystyle\int_0^1 x^2 dx = \frac{1}{3}$

Giải phương trình, hệ phương trình

Ví dụ 1.5. a) Phương trình $x^2 - 3x + 2 = 0$ có hai nghiệm là $x_1 = 1$ và $x_2 = 2$.

```
solve( x**2 - 3*x +2 , x)   # → [1, 2]
```

b) Hệ phương trình $\begin{cases} 3x - y = 5 \\ 2x - y = 2 \end{cases}$ có nghiệm $x = 3, y = 4$.

```
solve([3*x-y - 5, 2*x-y - 2], [x, y])   # → {x: 3, y: 4}
```

Giải phương trình vi phân

Ví dụ 1.6. Bài toán giá trị ban đầu $y' = y - x, y(0) = 2$ có nghiệm $y = e^x + x + 1$.

```
from sympy import *
x = symbols('x')
y = symbols('y', cls=Function)
dsolve(-y(x).diff() + y(x) - x, y(x), ics={y(0): 2})   #
```

$\rightarrow y(x) = x + e^x + 1$

1.5.7 Gói lệnh numpy

Gói lệnh hỗ trợ các phép toán đối với véctơ và ma trận. Khai báo bằng lệnh

```
import numpy as np
```

Véctơ

Đối với véctơ $x = (2, -1, -3)^T$:

Python	Kết quả	Mô tả
`x = [2, -1, -3]`		khai báo kiểu *dãy*
`x = np.array([2, -1, -3])`		khai báo thành kiểu *mảng* của gói `numpy`
`len(x)`	3	cỡ của x
`x[0]`	2	phần tử đầu của x

Với hai véctơ $x = (x_1, x_2, \dots, x_n)^T, y = (y_1, y_2, \dots, y_n)^T \in \mathbb{R}^n$ có kiểu mảng

Phép toán	Python	Ghi chú
$x + y = (x_1 + y_1, x_2 + y_2, \dots, x_n + y_n)$	`x + y`	
$x - y = (x_1 - y_1, x_2 - y_2, \dots, x_n - y_n)$	`x - y`	
$kx = (kx_1, kx_2, \dots, kx_n)$	`k * x`	$k \in \mathbb{R}$

$\langle x, y \rangle = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$ `x.dot(y)` hoặc `np.dot(x, y)` tích vô hướng

Lệnh `x.dot(y)` bắt buộc `x` có kiểu mảng, còn `np.dot(x, y)` với đối số kiểu dãy hay mảng đều dùng được.

Ma trận

Đối với ma trận $A = \begin{bmatrix} 3 & -1 & 2 \\ 0 & 4 & 1 \end{bmatrix}$:

Lệnh	Kết quả	Ý nghĩa
`A = [[3, -1, 2], [0, 4, 1]]`		khai báo kiểu *dãy* (lồng dãy)
`A = np.array( [[3, -1, 2], [0, 4, 1]] )`		khai báo kiểu *mảng*
`A.shape`	(2, 3)	số hàng và số cột của A
`A[0, 1]`	-1	hàng 0 cột 1 của A
`A.T`		chuyển vị của A
`np.identity(n)`	I_n	ma trận đơn vị cấp n
`len(A)`	2	số hàng của A
`A[0]`		hàng đầu tiên của A
`A[0][1]`	-1	hàng 0 cột 1 của A

Các lệnh ở khung cuối của bảng dùng được cho cả hai kiểu dữ liệu.

Với hai ma trận A, B có kiểu mảng, và cỡ tương thích (tức là, cỡ phù hợp để thực hiện được phép toán)

Phép toán	Python	Ghi chú
$A + B$	`A + B`	
$A - B$	`A - B`	
kA	`k * A`	$k \in \mathbb{R}$
AB	`A.dot(B)` hoặc `np.dot(A, B)`	
A^{-1}	`np.linalg.inv(A)`	

Lệnh `A.dot(B)` bắt buộc A có kiểu mảng.

Giải hệ phương trình tuyến tính

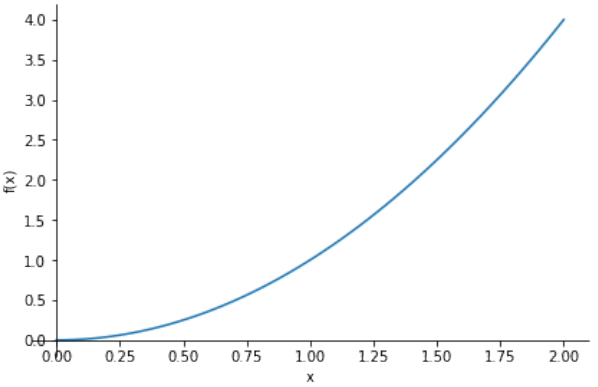
Ví dụ 1.7. Hệ phương trình tuyến tính trong Ví dụ 1.5(b), với nghiệm $x = 3, y = 4$, có thể viết dưới dạng ma trận

$$\begin{bmatrix} 3 & -1 \\ 2 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

```
A = [[3, -1], [2, -1]]
b = [5, 2]
np.linalg.solve(A, b)  # → array([3., 4.])
```

Vẽ đồ thị hàm số

Ví dụ 1.8. Vẽ đồ thị hàm số $f(x)$ trên đoạn $[0, 2]$.

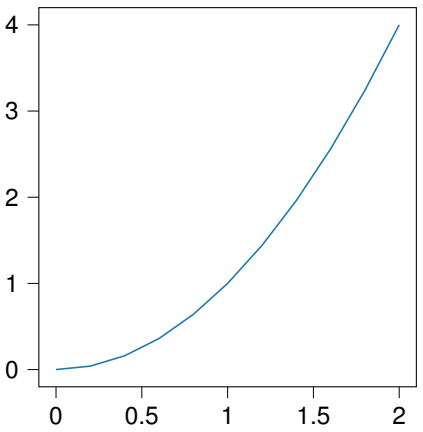



```
from sympy import *
x = symbols('x')
plot(x**2, (x, 0, 2))
```

1.5.8 Gói lệnh matplotlib: vẽ đồ thị

Đây là gói lệnh mở rộng các tính năng vẽ đồ thị.

Ví dụ 1.9. Vẽ đồ thị hàm số $y = x^2$ trên đoạn $[0, 2]$ bằng cách nối các điểm trên đồ thị, trong đó hoành độ các điểm chia đều đoạn $[0, 2]$ thành 10 đoạn bằng nhau.



```
import numpy as np
X = np.linspace(0, 2, 10 + 1)  # hoặc X = [0 + i * (2-0)/10 for
    i in range(10+1)]
Y = [x**2 for x in X]

import matplotlib.pyplot as plt
plt.plot(X, Y);
```

1.5.9 Cấu trúc điều khiển

if: rẽ nhánh

Trường hợp 1:

```
if 1 < 2:           # điều kiện
    print('đúng')   # thực hiện lệnh nếu điều kiện đúng
```

Trường hợp 2:

```
if 1 < 2:             # điều kiện
    print('đúng')     # thực hiện lệnh nếu điều kiện đúng
else:
    print('sai')      # thực hiện nếu điều kiện sai
```

for: lặp xác định

Ví dụ 1.10. In ra các số nguyên từ 0 tới 9, mỗi số một dòng.

Giải.

```
for i in range(10):   # i = 0,9
    print(i)
```

□

Ví dụ 1.11. In ra các biểu thức $x^5, x^6, \ldots, x^{10}$, mỗi biểu thức một dòng.

Giải. Để hiển thi

```
from sympy import *
x = symbols('x')

for i in range(5, 11):   # i = 5,10
    display(x**i)        # hiển thị công thức toán
```

□

while: lặp không xác định

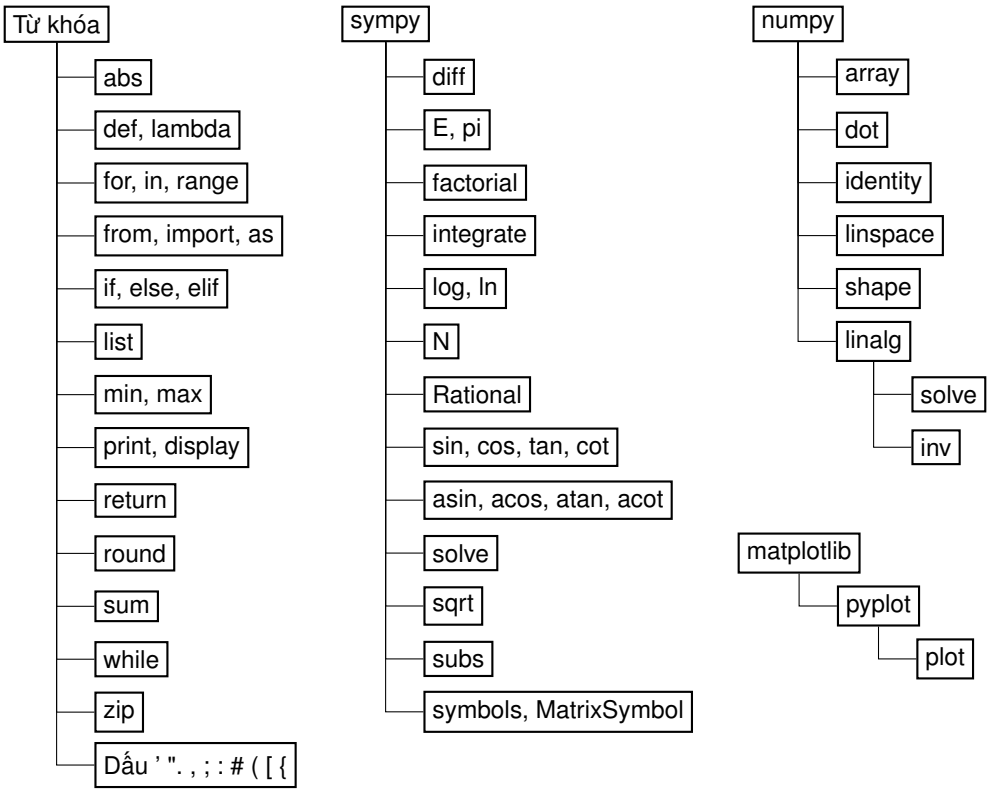
Ví dụ 1.12. Viết đoạn mã trong Ví dụ 1.10 dưới dạng vòng lặp không xác định.

Giải.

```
i = 0
while i < 10:      # lặp khi điều kiện còn đúng
    print(i)
    i += 1         # i = i + 1, có chức năng thay đổi giá trị của
    điều kiện
```

□

Tóm tắt Python



Tài liệu tham khảo

[1] Phạm Kỳ Anh. *Giải tích số*. Đại học Quốc gia Hà Nội, 2002. 284 trang.

[2] Richard L. Burden, Douglas J. Faires **and** Annette M. Burden. *Numerical Analysis*. phiên bản 10. Cengage Learning, 2016. 918 trang.

[3] NumPy community. *NumPy User Guide*. phiên bản 1.22.0. 531 trang. URL: https://numpy.org/doc/stable.

[4] SciPy community. *SciPy Reference Guide*. phiên bản 1.8.1. 3584 trang. URL: https://docs.scipy.org/doc.

[5] Phan Văn Hạp **and** Lê Đình Thịnh. *Phương pháp tính và các thuật toán*. Nhà xuất bản Giáo dục, 2000. 400 trang.

[6] Doãn Tam Hòe. *Toán học tính toán*. Đại học Quốc gia Hà Nội, 2009. 240 trang.

[7] Matplotlib development team. *Matplotlib documentation*. phiên bản 3.5.1. URL: https://matplotlib.org/3.5.1/tutorials/index.html.

[8] SymPy Development Team. *SymPy Documentation*. phiên bản 1.8. 2750 trang. URL: https://github.com/sympy/sympy/releases.