THỰC HÀNH TOÁN CAO CẤP

TÀI LIỆU PHỤC VỤ SINH VIÊN NGÀNH KHOA HỌC DỮ LIỆU

Nhóm biên soạn: TS. Hoàng Lê Minh – Khưu Minh Cảnh – Hoàng Thị Kiều Anh – Lê Thị Ngọc Huyên – …

TP.HCM – Năm 2019

MỤC LỤC

CHƯƠNG 1: XỬ LÝ SỐ VÀ HÌNH THỨC VỚI PYTHON

Mục tiêu:

- *Giới thiệu môn học*
- *Python như một máy tính siêu việt: các phép toán số học đơn giản với phần mềm Python*
- *Hàm số, tính chất và giới hạn của hàm số.*

Nội dung chính:

1. Giới thiệu về môn học và các yêu cầu

Phần này Giảng viên giới thiệu sơ lược về môn học Thực hành Toán Cao cấp để định hướng cho Sinh viên học tập và nghiên cứu có hiệu quả.

1.1. Về môn học Thực hành toán cao cấp

Theo "truyền thống", các môn toán nói chung và môn Toán cao cấp được giảng dạy cho sinh viên ở cấp Đại học gồm 2 phần chính:

- Một là: Lý thuyết và các chứng minh cho lý thuyết.
- Hai là: Các bài tập tính toán.

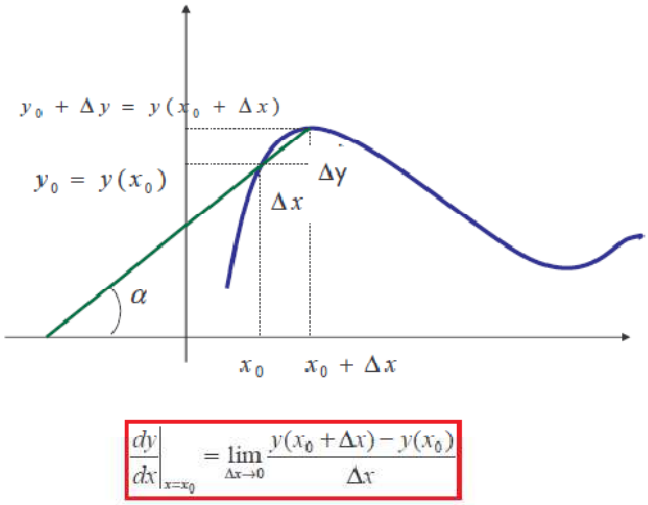
Đối với sinh viên, chiều hướng nắm rõ lý thuyết và hiểu rõ chứng minh để áp dụng giải bài tập tính toán là yêu cầu bắt buộc. Ngược lại, ở chiều hướng từ các số liệu thu thập thực tế để hình thành bài toán và giải, sinh viên cần có một tư duy hoặc (ít nhất) cần có khái niệm về một tư duy hoặc các ứng dụng thực tiễn được học để hình thành nên bài toán và tiếp cận phương pháp giải quyết. Môn học "Thực hành Toán Cao cấp" được xây dựng để sinh viên làm việc trên nền tảng xử lý của máy tính nhằm:

- Thể hiện, trình bày về lý thuyết và chứng minh toán học; trực quan hóa bằng các hình ảnh, biểu đồ;
- Áp dụng để xử lý các số liệu thực tế để làm rõ hơn cho lý thuyết;
- Vượt qua rào cản tính toán "bằng tay" để minh chứng sự đúng đắn của lý thuyết hoặc một chứng minh nào đó;
- Kỹ năng sử dụng phần mềm, đặc biệt phần mềm có tính kết nối cao với các lĩnh vực khác trong cuộc sống.

Với môn Toán Cao cấp (còn gọi là Giải tích), trên thực tế, năng lực của nền tảng tính toán, còn gọi là sức mạnh khả năng xử lý tính toán, trong thời kỳ Newton và Leibniz phát triển các lý thuyết vi tích phân bị giới hạn. Do đó, nhìn về lõi, hầu hết các ý tưởng cơ bản trong hướng Giải tích đều là những quá trình xấp xỉ (approximation), cụ thể là:

- Xử lý một chuỗi vô hạn bằng xấp xỉ tổng…;

- Tính đạo hàm bằng xấp xỉ các đường…;
- Định nghĩa tích phân bằng xấp xỉ tổng….;
- Sử dụng phương pháp Euler để giải phương trình vi phân cũng là một phương pháp xấp xỉ….;



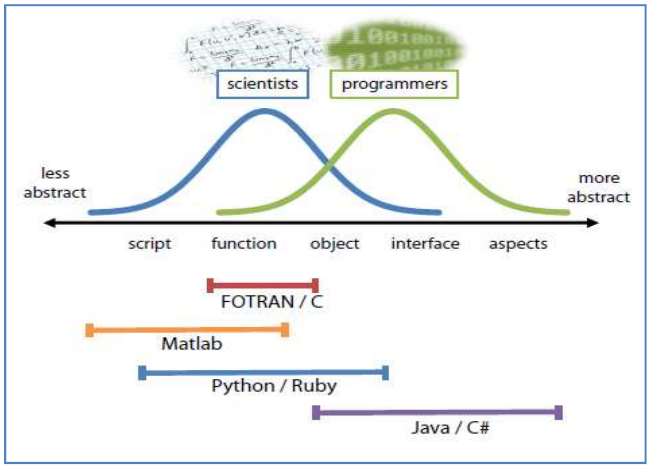
Hình 1: Mô tả về ý nghĩa của đạo hàm bằng thể hiện hình học

Từ đó cho thấy, một trong những mục tiêu chính của Sinh viên cần tiếp thu là *vấn đề giới hạn của những tính toán xấp xỉ* đó (the limit of such approximations).

Với môn học *Thực hành Toán cao cấp* này, ngôn ngữ Python được chọn thay cho các ngôn ngữ khác như Maple, Mathematica, Matlab với 5 ưu điểm như sau:

- Một là: Ngôn ngữ có tính mở và không phải là phần mềm chuyên dụng cho toán học! ← nhằm hạn chế tư duy thụ động, chỉ cần áp dụng công thức, nghĩa là không những phải học công cụ để sử dụng (tính toán) mà còn phải học bản chất của vấn đề. Sinh viên sẽ có điều kiện hơn để nắm vững cách thức tính toán cụ thể thay vì chỉ học lệnh thuần túy.
- Hai là: Cấu trúc đơn giản, dễ hiểu, dễ học.
- Ba là: Thư viện toán học đầy đủ và mạnh mẽ.
- Bốn là: Mã nguồn mở.
- Năm là: Được sử dụng rộng rãi trong các lĩnh vực ứng dụng thực tiễn.

Trên thực tế, Python được nhiều người sử dụng và biến đến cho đến ngày nay. Dưới đây là hình vẽ về Python với các chức năng sử dụng:



Hình 2: Python là ngôn ngữ mạnh về các mảng: lập trình script, hàm và đối tượng

Hình ảnh trên cho thấy vị trí của các ngôn ngữ. Các ngôn ngữ luôn cố gắng hoàn thiện hơn để trở thành ngôn ngữ nhiều người sử dụng. Hiện nay, với sự hỗ trợ của cộng đồng sử dụng lớn, Python đã có thể "lập trình hướng khía cạnh" (Aspect Oriented Programming, viết tắt là AOP) và đang phát triển mạnh mẽ về hướng hỗ trợ các nhà khoa học. Lưu ý các thuật ngữ khác:

- Lập trình script: là lập trình các đoạn lệnh.
- Lập trình function: là có khả năng lập ra các hàm và triệu gọi các hàm.
- Lập trình object:là khả năng xây dựng các đối tượng. Hiện tại, đa số các ngôn ngữ lập trình thế hệ mới đều có khả năng lập trình hướng đối tượng nhằm tạo ra các đối tượng để việc kế thừa sử dụng được nhanh chóng, thuận tiện và thúc đẩy phát triển phần mềm: ít lỗi hơn, giảm các chi phí xây dựng phần mềm, các đối tượng có thể xây dựng độc lập,…

Thực hành 1: Hãy viết lệnh tính tổng chuỗi sau:

$$S = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots$$

Hướng dẫn thực hành:

- *Sinh viên thực hiện việc mở trình Python trên máy.*
- *Sinh viên đánh lệnh trong phần chi tiết thực hành và báo cáo kết quả.*

Chi tiết thực hành:

Về mặt toán học, tổng của chuỗi trên sẽ là tổng của những số có dạng $\frac{1}{2^k}, k \geq 0, k \in \mathbb{N}$. [Giảng viên có thể giải thích với Sinh viên bằng hình ảnh về tổng trên]

Từ dấu nhắc của Python >>>, Sinh viên hãy đánh vào các lệnh sau và cho biết ý nghĩa.

Lưu ý: thay đổi giá trị biến blocks lần lượt bằng 3, 5 và 10000

```
>>> blocks = 10000
>>> dayS = [1/2**n for n in range(0, blocks)]
>>> tongS = sum(dayS)
>>> print (tongS)
```

…… ← *Sinh viên cho biết kết quả và giải thích từng lệnh trên.*

Thực hành 2: Tính tích phân sau **theo định nghĩa**:

$$I = \int_0^2 x^2 dx$$

Hướng dẫn: Giảng viên gọi Sinh viên lên bảng giải bài toán và nêu ý nghĩa.

Theo định nghĩa, đoạn [0,2] sẽ được chia nhỏ thành $n + 1$ giá trị (từ 0 đến 2). Giá trị tích phân trên sẽ là tổng của $n + 1$ số hạng:

$$S_k = x_k^2 \times (x_k - x_{k-1})$$

$(x_k - x_{k-1})$ đóng vai trò là dx. Hiển nhiên, càng chia nhỏ thì giá trị tổng (tích phân) càng chính xác! [Giảng viên có thể vẽ đồ thị hàm x^2 mô tả bằng hình học về cách tính trên bảng]

Sau đó, Sinh viên thực hiện các lệnh sau và cho kết quả:

```
>>> n = 1001
>>> X = [k/(n/2) for k in range(0, n)]
>>> S = 0
>>> for k in range(1, n):
        S = S + X[k]**2 * (X[k] - X[k-1])
>>> print (S)
```

…… ← *Sinh viên cho biết kết quả và giải thích từng lệnh trên.*

1.2. Những yêu cầu đối với sinh viên

Sinh viên được yêu cầu đáp ứng những vấn đề sau với môn học:

- Đi học đầy đủ, khuyến khích tham gia làm bài tập trên lớp;
- Thi giữa kỳ và cuối kỳ đầy đủ;

- Làm bài tập nhà khi được yêu cầu đầy đủ.
- Học các môn và kiến thức liên quan (Toán cao cấp, lập trình Python).
- Và những yêu cầu khác theo quy định của Nhà trường, Khoa và Phòng máy.

2. Giới thiệu môi trường làm việc Python

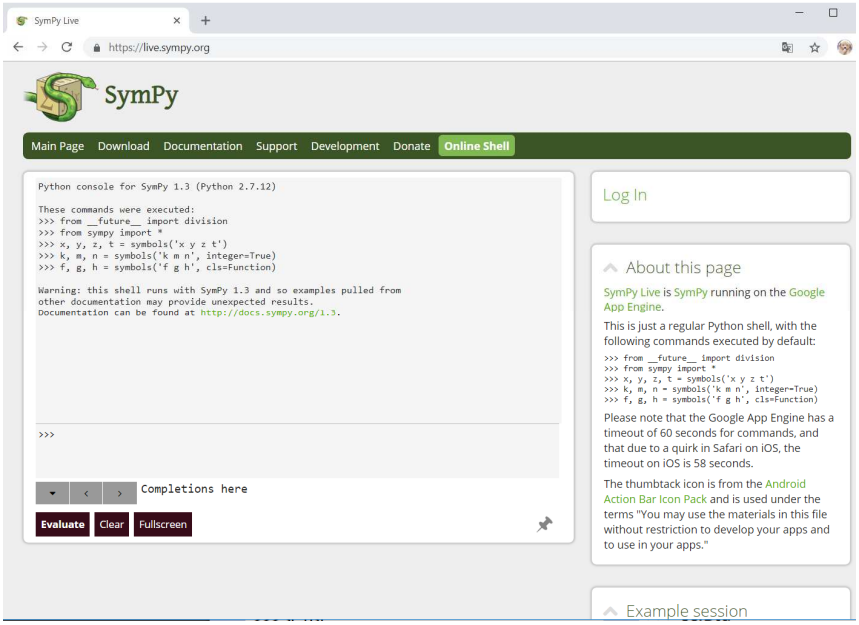
Hiện nay, Python là một ngôn ngữ lập trình được nhiều người sử dụng và nhiều ứng dụng lớn trên thế giới, đặc biệt là các ứng dụng web của Google, Amazon,... [Giảng viên có thể hướng dẫn để Sinh viên có thể tham khảo thêm thông tin trên Google và Wikipedia về Python].

Với triết lý đơn giản, hiện tại, Python được nhiều ngành nghề sử dụng như một công cụ giao tiếp với máy tính để tính toán như: địa lý,... Python hỗ trợ mạnh mẽ các công cụ tính toán với các hàm thư viện toán học về: giải tích, đại số, thống kê, hình học,… Hiện tại, Python có 2 nhánh chính là 2.x và 3.x. Nhánh 2.x kế thừa tính lâu đời và truyền thống của ngôn ngữ Python. Nhánh 3.x nhiều tính năng hiện đại được tích hợp và ngày càng lớn mạnh. Dự kiến 5-10 năm nữa, các hệ thống sử dụng Python 2.x sẽ được chuyển sang nhánh Python 3.x.

Hiện nay, Python có gói phần mềm đóng gói sẵn để thuận tiện xử lý theo các yêu cầu. Mỗi gói phần mềm được tích hợp chọn lọc nhiều gói thư viện được "gắn" vào trong phần lõi chính. Sympy trực tuyến hoặc Anaconda là một trong những gói phần mềm gồm nhiều thư viện Python hỗ trợ cho việc tính toán.

2.1. Giới thiệu Sympy và cơ bản sử dụng Sympy

2.2. Sử dụng Python trực tuyến với gói live sympy và sympy trên Ananconda



Hình 3: Màn hình trang web https://live.sympy.org

Trong bài thực hành này, Sinh viên sẽ được tiếp cận để làm quen khả năng tính toán cơ bản của Python bằng việc sử dụng một hệ Python trực tuyến. Theo đó, với những nơi có điều kiện về mạng Internet, chúng ta có thể trãi nghiệm gói SymPy trực tuyến bằng việc truy cập trang web: http://live.sympy.org. Đây là hệ tính toán bằng gói SymPy trực tuyến trên lõi Python phiên bản 2.7. Giao diện trang web gồm: khung giữa là nơi thực thi các lệnh với dấu nhắc >>>; bên phải là giới thiệu (About this page); và phía dưới là các minh họa ngẫu nhiên được giới thiệu (Example session). *Lưu ý: Với các bài thực hành trong các chương sau, gói Anaconda 3 sẽ được sử dụng.*

Ngoài ra, với gói Anaconda 3 cài đặt trên máy, để sử dụng, chúng ta thực hiện việc đưa thư viện **sympy** vào trong hệ thống Python như sau:

```
from sympy import *
```

2.3. Cơ bản về SymPy

2.3.1. Các lệnh cơ bản trong SymPy

SymPy định nghĩa 3 dạng dữ liệu: số thực (Real), phân số (Rational) và số nguyên (Integer). Phân số là thể hiện tỉ số giữa cặp số nguyên. Ví dụ: $\frac{5}{2}$ được thể hiện là Rational(5, 2)… Cụ thể:

```
>>> from sympy import *
>>> a = Rational(1,2)

>>> a
1/2

>>> a*2
1
```

Lưu ý: lệnh import này sẽ đưa thư viện sympy vào sử dụng. Từ đó, chúng ta không cần phải nhắc đến gói sympy ở các lệnh như sympy.___ như khi sử dụng lệnh ***import sympy*** bên trên.

Bên cạnh đó, SymPy sử dụng nền tảng mpmath nên SymPy có hỗ trợ các giá trị hằng số như số *e* (kí hiệu là chữ e), số π (kí hiệu là chữ pi) và số vô cùng ∞ (kí hiệu là 2 chữ o nhỏ: oo). Minh họa: Số vô cùng.

```
>>> oo > 99999
True
>>> oo + 1
oo
```

*Lưu ý: Chúng ta phải sử dụng lệnh **sympy.oo** thay vì **oo** trong minh họa trên nếu như phía trên đó, chúng ta sử dụng **import sympy** (thay vì lệnh **from sympy import ***).*

Và minh họa về số π và số e:

```
>>> pi**2
pi**2

>>> pi.evalf()
3.14159265358979

>>> (pi+exp(1)).evalf()
5.85987448204884
```

Thực hành 3: Thử nghiệm tính toán sau:

>>> **Rational**(1, 2) + **Rational**(1, 3)

…… ← sinh viên điền kết quả

>>> **Rational**(1, 2) + **1**

…… ← sinh viên điền kết quả

>>> **Rational**(1, 3) + **1** + **1.5**

…… ← sinh viên điền kết quả

2.3.2. Thực hành khai báo biến để sử dụng

Gói SymPy bắt buộc người sử dụng khai báo "biến" (kí hiệu toán học, symbol) của hàm trước khi sử dụng. Ví dụ:

Thực hành 4: Về khai báo biến trong Sympy:

```
>>> from sympy import *
>>> x = Symbol('x')
>>> y = Symbol('y')
```

Sau đó, chúng ta có thể đưa vào phương trình như:

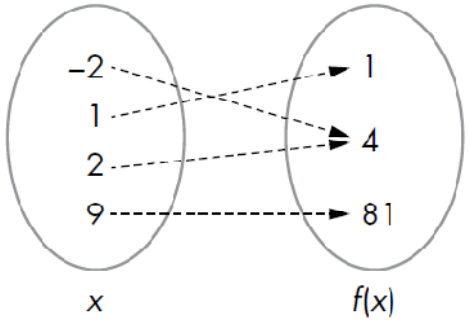
```
>>> x+y+x-y
2*x

>>> (x+y)**2
(x + y)**2
```

Với các kí hiệu toán học, các phép toán được áp dụng là: (nhóm số học) +, -, *, **; (nhóm luận lý) &, |, ~, >>, <<.

3. Hàm số, tính chất và giới hạn của hàm số

Chúng ta bắt đầu với một số định nghĩa cơ bản về hàm số. Hàm số được xem là ánh xạ giữa tập đầu vào (input set) và tập đầu ra (output set). *Điểm yêu cầu chính của hàm số là mỗi phần tử tập đầu vào chỉ duy nhất một phần tử trong tập đầu ra.*

Ví dụ: Hàm số $f(x) = x^2$ được thể hiện như sau:



Kí hiệu $f(x) = x^2$ với x được gọi là biến số độc lập với giá trị thuộc một miền xác định (domain) và hàm f có 1 biến là x.

Ngoài ra, chúng ta có các hàm đa biến (nhiều biến). Ví dụ: hàm 2 biến x và y như sau:

$$f(x, y) = x^2 + y^2$$

3.1. Miền xác định và miền giá trị của hàm

Miền xác định (domain) của hàm là tập các giá trị đầu vào (input set) của các biến số hợp lệ. Miền giá trị (range) của hàm là tập đầu ra (output set).

Ví dụ miền xác định cho hàm dạng thương số: Xét hàm $f(x) = \frac{1}{x}$ không thể có giá trị bằng 0 vì $\frac{1}{0}$ không được định nghĩa. Miền giá trị của được xác định bằng giá trị của hàm với các giá trị trong miền xác định. Từ đó, hàm trên sẽ có miền giá trị là tập số thực khác 0. Với một số hàm số thực hiện phép chia (thương), các điểm loại trừ là những điểm làm giá trị mẫu số bằng 0.

SymPy cung cấp phương thức denoms trong gói **sympy.solvers.solvers** để chỉ ra tập các biểu thức có thể bằng 0 trong thương số:

Thực hành 5a: Tìm các hàm ở mẫu số của biểu thức sau: $\frac{1}{x} \times \frac{1}{x-3}$

```
>>> from sympy.solvers.solvers import denoms

>>> eq = (1/x)*1/(x-3)

>>> dd = denoms(eq)

>>> print (dd)
```

…… ← sinh viên tự ghi kết quả.

Từ đó, chúng ta có thể "giải" (solve) các phương trình trong tập đó để tìm nghiệm.

Thực hành 5b: Với $f = \frac{\left(1+\frac{1}{x}\right)}{x-1}$. Hãy tìm các điểm hàm số không xác định (tập điểm loại trừ).

```
>>> eq = (1+1/x)/(x-1)
>>> from sympy.solvers.solvers import denoms
>>> loai_tru = set()
>>> for d in denoms(eq):
        for s in solve(d):
            loai_tru.add(s)
>>> print (loai_tru)
```

…… ← Sinh viên điền vào

Biểu thức eq mô tả hàm $f = \frac{\left(1+\frac{1}{x}\right)}{x-1}$ có hai điểm loại trừ đó là 0 và 1. Hay nói cách khác f không xác định tại $\{0, 1\}$.

3.2. Lập giả thuyết toán học trong Sympy

Trong chương trình trên, chúng ta tạo ra đối tượng Symbol (của SymPy) bằng việc định nghĩa biến >>> x = Symbol('x'). *Lưu ý: để khai báo nhiều biến cùng lúc, chúng ta sử dụng symbols như sau: x,y,z = symbols('x y z').*

Chúng ta xét mô tả sau để xem sự hoạt động của SymPy. Bài toán giả định là: ***"SymPy liệu có thể kiểm tra biểu thức x+3 dương được không?"***. Chúng ta hãy xét các lệnh thử nghiệm về tính "thông minh" của SymPy như sau:

Thực hành 6: Lập "giả thuyết" trong Sympy

```
>>> from sympy import Symbol
>>> x = Symbol('x')
>>> if (x+3) > 0:
        print('Chac chan x+3 duong!')  # nếu tổng này là số dương thì in ra câu này
else:
```

print ('x+3 chua chac la so duong!') # ngược lại thì in ra câu này

Kết quả của chương trình thực thi:

```
>>> from sympy import Symbol
>>> x = Symbol('x')
>>> if (x+3) > 0:
        print('Chac chan x+3 duong!')
else:
        print ('x+3 chua chac la so duong!')


Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    if (x+3) > 0:
  File "C:\Anaconda3\lib\site-packages\sympy\core\relational.py",
line 229, in __nonzero__
    raise TypeError("cannot determine truth value of Relational")
TypeError: cannot determine truth value of Relational
>>>
```

Lỗi trả về là không thể kết luận được câu chuyện (x+3) là số dương hay không. Lí do được giải thích là với một kí hiệu **x** mới được khai báo thì việc kết luận giá trị x+3 > 0 là điều không thể và Python đã trả về lỗi **TypeError** (lỗi về loại) mà cụ thể là không thể xác định được **chân trị** (truth value) của quan hệ (if).

Tuy nhiên, trong toán học, chúng ta đều biết rằng, nếu đã biết trước x là một số dương, thì suy luận x+3 > 0 là một điều hiển nhiên, ngược lại, nếu x là số âm thì chúng ta cần phải xem xét kỹ hơn giá trị x để kết luận dấu của biểu thức (x+3).

Đối tượng Symbol cho phép chúng ta xác định x là số dương dựa trên tham số positive=True khi khai báo đối tượng. Với giả thuyết là số dương, chúng ta sẽ có biểu thức (x+3) luôn dương. Minh họa bằng đoạn lệnh như sau:

>>> x = Symbol('x', positive = True)

>>> if (x+3) > 0:

print('Chac chan x+3 duong!')

else:

print ('x+3 chua chac la so duong!')

Sinh viên hãy cho biết kết quả: …….

```
>>> x = Symbol('x', positive = True)
>>> if (x+3) > 0:
        print('Chac chan x+3 duong!')
else:
        print ('x+3 chua chac la so duong!')


Chac chan x+3 duong!
>>>
```

Lưu ý 1: khi chúng ta xác định x là số âm (negative=True), chúng ta sẽ nhận được kết quả lỗi như lúc chúng ta chưa xác định giá trị x. Ví dụ:

>>> x = Symbol('x', negative = True)

>>> if (x+3) > 0:

print('Chac chan x+3 duong!')

else:

print ('x+3 chua chac la so duong!')

Lưu ý 2: Ngoài positive và negative, các dạng mô tả đối tượng Symbol được hỗ trợ bao gồm: real, integer, complex, imaginary,… Những khai báo đó như những giả thuyết toán học trong SymPy được môi trường Python xử lý.

Ví dụ:

```
>>> x = Symbol('x', integer = True)
>>> y = Symbol('x', real = True)
>>> z = Symbol('x', complex = True)
```

3.3. Các hàm toán học sơ cấp

Các hàm toán học thông thường được cung cấp từ thư viện **math**, là thư viện chuẩn của Python. Ví dụ các hàm sin(), cos() là những hàm lượng giác trả về giá trị sin và cosin. Ngoài ra, thư viện định nghĩa các hàm khác như tan(),… và những hàm tính ngược như asin(), acos() cũng như atan()…

Module math cũng tính toán được các hàm log, cụ thể: log() là hàm tính log cơ số tự nhiên, log2() là hàm tính log cơ số 2, tương tự log10(),… Và tương ứng là các hàm mũ exp() để tính giá trị e^xvới e là số Euler (khoảng 2.71828).

Nhược điểm của các hàm được thư viện **math** hỗ trợ là không phù hợp với việc *xử lý biểu thức hình thức* (symbolic expression). Nếu chúng ta muốn xử lý những phép toán đại số với các kí hiệu, chúng ta phải sử dụng gói thư viện SymPy.

Xét các ví dụ dưới đây:

Thực hành 7: Xử lý biến trong Sympy

```
>>> import math
>>> math.sin(math.pi/2)
1.0
```

Trong ví dụ trên, chúng ta thấy rằng với góc $\pi/2$ được thể hiện là **math.pi/2** và chúng ta dễ dàng được tính toán giá trị sin của nó. Dưới đây, chúng ta sử dụng gói thư viện SymPy như sau:

```
>>> import sympy
>>> sympy.sin(math.pi/2)
1.00000000000000
```

Như vậy, tương tự với hàm **sin()** của gói thư viện chuẩn (thư viện math) và hàm sin() của SymPy có tham số cần nhập là một góc theo giá trị **radian**.

Bây giờ, thay vì nhập bằng số, chúng ta thử gọi hàm với một biến biểu tượng (symbol):

```
>>> from sympy import Symbol
>>> goc_theta = Symbol("theta")
>>> import math
>>> math.sin(goc_theta) + math.sin(goc_theta)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    math.sin(goc_theta) + math.sin(goc_theta)
  File "C:\Anaconda3\lib\site-packages\sympy\core\expr.py", line 256, in __float
__
    raise TypeError("can't convert expression to float")
TypeError: can't convert expression to float
>>> import sympy
>>> sympy.sin(goc_theta) + sympy.sin(goc_theta)
2*sin(theta)
>>>
```

Hàm sin() của thư viện chuẩn (math) không hiểu cần làm điều gì khi chúng ta gọi xử lý một đối tượng có kiểu là Symbol. Do đó, nó sẽ phát sinh ra lỗi và đưa ra hướng dẫn rằng nó cần một giá trị số trong tham số của hàm **sin()**. Ngược lại, SymPy thực hiện việc tính toán và trả về kết quả là giá trị 2*sin(theta). Điều này cho thấy sự thú vị về xử lý toán học trong SymPy. Tuy nhiên, đó cũng là sự mô tả về loại tác vụ mà các hàm toán học trong thư viện chuẩn (gói math) không thể đáp ứng được khi xử lý về hình thức.

Chúng ta có thể thử lại:

>>> 2*sympy.sin(theta) == sympy.sin(theta) + sympy.sin(theta)

…… ← Sinh viên ghi lại kết quả so sánh bằng của 2 biểu thức.

Tuy nhiên, chúng ta lưu ý rằng, để so sánh thực sự các biểu thức, Sympy cung cấp phương thức equals() thay cho toán tử '==’. Ví dụ: “chứng minh” biểu thức $sin2x = 2sinxcosx$ bằng Sympy:

>>> (2*sympy.sin(theta)*sympy.cos(theta)).**equals**(sympy.sin(2*theta))

…… ← Sinh viên ghi lại kết quả

Xét qua một ví dụ khác, xét biểu thức có biến thời gian t cho một vật thể chuyển động của viên đạn đại bác khi được bắn lên trời (projectile motion) để đạt đến đỉnh cao nhất khi được ném với vận tốc ban đầu là u và ở một góc $theta$.

Tại điểm cao nhất, ta có phương trình $u\sin(theta) - gt = 0$. Để tìm t, chúng ta sử dụng hàm solve() của thư viên SymPy như sau:

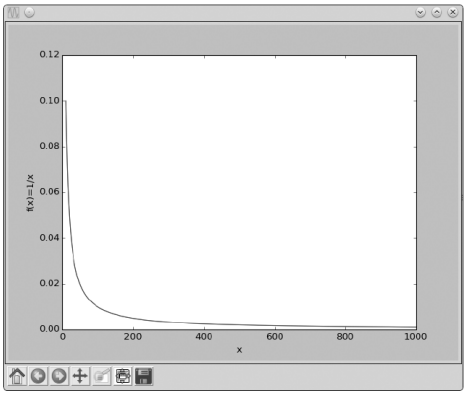
Thực hành 8: Giải phương trình đơn giản bằng sympy

```
>>> from sympy import sin, solve, Symbol
>>> u = Symbol("u")
>>> t = Symbol("t")
>>> g = Symbol("g")
>>> theta = Symbol('theta')
>>> solve(u*sin(theta)-g*t, t)
[u*sin(theta)/g]
>>>
```

Như vậy, chúng ta dễ dàng xác định được biểu thức tính toán t là **u*sin(theta)/g** với hàm **solve**(). Như vậy hàm **solve**() ở đây sẽ giải phương trình mặc định vế phải bằng 0 và biến được sử dụng là t.

3.4. Giới hạn của hàm số

Bài toán thường thấy trong giải tích là tìm giá trị giới hạn (limit hay gọi ngắn gọn là **lim**) của một hàm số khi biến số được giả định tiến đến một giá trị nào đó (giá trị xác định, hoặc vô cùng).



Xét hàm số $f(x) = {}^{1}/_{x}$ (với biểu đồ như hình bên dưới).

Khi giá trị x tăng, giá trị của hàm $f(x)$ sẽ dần về 0. Kí hiệu toán học là:

$$\lim_{x\to\infty} \frac{1}{x} = 0$$

Với SymPy, chúng ta có thể tính giới hạn của hàm số bằng lớp **Limit** như sau:

Thực hành 9: Tìm giới hạn của hàm số bằng Sympy

```
>>> from sympy import Limit, Symbol, S
>>> x = Symbol('x')
>>> Limit(1/x, x, S.Infinity)
Limit(1/x, x, oo, dir='-')
>>> gioihan = Limit(1/x, x, S.Infinity)
>>> gioihan.doit()
0
>>>
```

Giải thích:

- Dòng lệnh 1: Tham chiếu các thư viện, bao gồm thư viện lớp tính giới hạn, biểu tượng và lớp đặc biệt S chứa các định nghĩa giá trị vô cùng (dương vô cùng và âm vô cùng).
- Dòng lệnh 2: Tạo ra đối tượng biến x.
- Dòng lệnh 3: Tạo đối tượng giới hạn với 3 tham số: hàm số, biến của hàm và biến tính giới hạn (là giá trị vô cùng được biểu diễn bằng S.Infinity).
  Khi thực hiện dòng lệnh này, đối tượng được tạo ra trong bộ nhớ. Tuy vậy, để sử dụng đối tượng, chúng ta cần có một đối tượng (của chương trình Python máy tính) để lưu giữ và xử lý. Hiển nhiên đối tượng này có kiểu là Limit.
- Dòng lệnh 4: Viết lại dòng lệnh 3 và có phần lưu trữ đối tượng khai báo giới hạn trong một đối tượng (của chương trình Python máy tính) tên là 'gioihan'.
- Dòng lệnh 5: Thực hiện việc tính toán trên đối tượng 'gioihan' bằng việc gọi phương thức doit() được kế thừa từ đối tượng Limit của Sympy. Thực hiện lệnh nghĩa là chúng ta tính toán được giá trị giới hạn.

Mặc định giới hạn được xác định là hướng chiều dương. Tuy nhiên, nhiều trường hợp, chúng ta phải xác định hướng tiến đến của biến giới hạn vì kết quả giới hạn sẽ là âm hoặc dương vô cùng tùy theo hướng. Sympy cung cấp cho chúng ta từ khóa **dir** để mô tả hướng của giới hạn như sau:

Thực hành 10a: Tính toán giới hạn trái phải

>>> from sympy import **Limit**

>>> Limit(1/x, x, 0, dir='–').doit()

…….. ← Sinh viên thực hiện và điền kết quả.

>>> Limit(1/x, x, 0, dir='+').doit()

…….. ← Sinh viên thực hiện và điền kết quả.

Ngoài ra, lớp **Limit** có thể xử lý các hàm bất định, dạng 0/0 hoặc vô cùng/vô cùng một cách tự động. Đó là những giới hạn trong lý thuyết được tính toán bằng quy tắc l'Hôpital.

Xét ví dụ sau tính giới hạn của:

$$\lim_{x \to 0} \frac{\sin(x)}{x}$$

Thực hành 10b: Tính toán giới hạn dạng vô cùng/vô cùng

>>> from sympy import Symbol, sin

>>> Limit(sin(x)/x, x, 0).doit()

……. ← Sinh viên điền kết quả

Thực hành 11a: Tính giá trị của giới hạn sau khi x tiến đến vô cực:

$$x sin \frac{1}{x}$$

Lưu ý: giá trị vô cực là giá trị S. Infinity

Thực hành 11b: Tính toán giới hạn

>>> from sympy import limit, sin, S

>>> limit(x*sin(1/x), x, S.Infinity)

…… ← sinh viên điền kết quả

Hoặc:

>>> from sympy import limit, sin, S

>>> limit(x*sin(1/x), x, **sympy.oo**)

4. Một số ứng dụng

Phần ứng dụng này khuyến khích sinh viên đọc thêm trên lớp và về nhà làm thêm để hiểu bài.

4.1. Bài toán lãi suất kép liên tục – Continous Compound Interest

Giả định chúng ta có vốn 1 triệu đô trong ngân hàng. Và tổng tiền chúng ta có được trong 1 năm theo với lãi suất 100% nhận được từ n lần trong 1 năm là:

$$A = \left(1 + \frac{1}{n}\right)^n$$

Nhà toán học James Bernoulli khám phá ra rằng khi n tăng, số hạng $(1 + 1/n)^n$ tiến đến số e, là một giá trị hằng. Chúng ta có thể kiểm tra bằng cách tìm giới hạn của hàm số với các lệnh Python:

```
>>> from sympy import Limit, Symbol, S
>>> n = Symbol('n')
>>> Limit((1+1/n)**n, n, S.Infinity).doit()
E
```

Với giá trị vốn là p, lãi suất là r và số năm là t thì số tiền chúng ta

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

Giả định lãi suất liên tục, chúng ta có thể tìm được công thức cho giá trị A như sau:

Thực hành 12: Tìm công thức từ giới hạn của hàm số

```
>>> from sympy import Symbol, Limit, S
>>> p = Symbol('p', positive=True)
>>> r = Symbol('r', positive=True)
>>> t = Symbol('t', positive=True)
>>> Limit(p*(1+r/n)**(n*t), n, S.Infinity).doit()
p*exp(r*t)
```

Ở đây, chúng ta tạo 3 đối tượng thể hiện: vốn p, lãi suất r và số năm là t. Chúng ta cũng báo với SymPy là các biến này đều dương (với từ khóa positive=True). Lưu ý: nếu chúng ta không báo, SymPy sẽ không biết các giá trị số và không thể tính các giá trị một cách chính xác [Sinh viên có thể thử nghiệm]. Lệnh cuối là ta thực hiện việc tính giới hạn và tính toán cụ thể. Kết quả của tính toán cụ thể bằng phương thức **doit()** là biểu thức $p * \exp(r * t)$ cho chúng ta biết được lãi suất sẽ gia tăng theo hàm mũ theo thời gian với một lãi suất cố định.

4.2. Tỉ lệ thay đổi tức thời

Giả định một chiếc xe đang chạy trên đường tăng tốc đều đều trong suốt quãng đường S được mô tả bởi hàm số sau:

$$S(t) = 5t^2 + 2\mathrm{t} + 8$$

Trong hàm này, biến độc lập là biến t thể hiện thời gian di chuyển của chiếc xe từ lúc xe di chuyển.

Nếu chúng ta đo khoảng cách di chuyển trong khoảng thời gian từ thời điểm t_1đến thời điểm t_2 ($t_1 < t_2$) thì chúng ta có thể tính được khoảng cách xe đã di chuyển trong một đơn vị thời gian, thể hiện bằng biểu thức:

$$\frac{S(t_2) - S(t_1)}{t_2 - t_1}$$

Điều trên nói về tỉ lệ thay đổi trung bình của hàm $S(t)$ theo biến t, nói cách khác, đó là vận tốc trung bình. Nếu chúng ta viết $t_2 = t_1 + \delta_t$ với δ_t là khoảng thời gian giữa t_2 và t_1, khi đó, chúng ta có thể viết lại biểu thức trên như sau:

$$\frac{S(t_1 + \delta_t) - S(t_1)}{\delta_t}$$

Biểu thức trên có một biến là t_1. Giả định với δ_t rất nhỏ, gần như tiến đến 0, chúng ta có thể sử dụng kí hiệu giới hạn như sau:

$$\lim_{\delta_t \to 0} \frac{S(t_1 + \delta_t) - S(t_1)}{\delta_t}$$

Chúng ta sẽ tính toán giới hạn trên. Đầu tiên, chúng ta tạo ra các đối tượng:

Thực hành 13: Sử dụng lệnh **subs** để thay thế

```
>>> from sympy import Symbol, Limit
>>> t = Symbol('t')
>>> St = 5*t**2 + 2*t + 8

>>> t1 = Symbol('t1')
>>> delta_t = Symbol('delta_t')

>>> St1 = St.subs({t: t1})
>>> St1_delta = St.subs({t: t1 + delta_t})
```

Diễn giải các câu lệnh trên:

- Dòng 1: nạp thư viện. Chúng ta sử dụng 2 thư viện là Symbol và Limit.
- Dòng 2: khai báo biểu tượng/biến t.
- Dòng 3: định nghĩa hàm $S(t)$ hàm S theo biến t.
- Dòng 4 và 5: định nghĩa hai biến là t1 và delta_t tương ứng với kí hiệu toán học t_1 và δ_t.
- Dòng 6 và 7: Sử dụng phương thức **subs()** để thay thế giá trị $S(t_1)$ và $S(t_1 + \delta_t)$

Bây giờ, chúng ta có thể tính giá trị giới hạn:

```
>>> Limit((St1_delta-St1)/delta_t, delta_t, 0).doit()
10*t1 + 2
```

Giá trị trả về của giới hạn là $10 * t1 + 2$ mang ý nghĩa là tốc độ thay đổi của $S(t)$ tại thời điểm $t1$, hoặc đó là tỉ lệ thay đổi thức thời. Sự thay đổi này có nghĩa là vận tốc tức thời của xe tại thời điểm $t1$.

Giới hạn tính ở đây có ý nghĩa như là **đạo hàm** của hàm. Và chúng ta có thể tính toán nó trực tiếp nó thông qua lớp **Derivative** của Sympy!

Trong chương sau, Sinh viên sẽ được giới thiệu về xử lý đạo hàm bên cạnh những kỹ thuật xử lý toán học cơ bản khác cũng như vẽ biểu đồ với Sympy, cùng các ứng dụng lý thú khác.

BÀI TẬP CHƯƠNG 1

Những bài tập này được yêu cầu sinh viên nộp bài trong tuần học kế tiếp

Bài tập 1: Sai số và số có ý nghĩa trong Python

Nói về Python, chúng ta phải xem xét đến "tính chất" của một ngôn ngữ lập trình. Các biểu thức tính toán và luận lý trong Python cũng phải tuân theo mức độ mô hình hóa kiểu dữ liệu.

Ví dụ: Sinh viên thử nghiệm điều "vô lý" trong toán học sau trong môi trường Python:

```
>>> 3 == 3
True
>>> 3.000000000000000000000000000000000001 == 3.000000000000000000000000
True
>>> 3.00000000000000001 == 3.00000000000000000
True
```

Bài thực hành: Tìm số lượng số 0 để biểu thức 3.00…001 == 3.00..000 mang giá trị True!

Đoạn code Python dưới đây sẽ tìm số lượng số 0 để phép so sánh bên trên mang giá trị True.

Trước hết, sinh viên hãy thử nghiệm các lệnh sau để xem kết quả:

```
>>> 3.1 == 3.0
False
>>> 3.01 == 3.0
False
>>> 3.001 == 3.0
False
```

Sau đó, sinh viên hãy thực hiện và giải thích đoạn mã bên dưới và cho biết giá trị i tìm được là bao nhiêu?

```
>>> a = 3.1; b=3.0; i = -1
>>> while (a != b):
        print a, " != ", b
        a_new = a - 0.9 * 10 ** (i)
        if (a_new != a):
                a = a_new
                i = i - 1
        else:
                print i
                a = b
```

……

……

……..

……

……

……

i =……? Nghĩa là có bao nhiêu số các số 0 thì giá trị phía sau sẽ là không có ý nghĩa (3.000..0001), tương ứng với giá trị 10^i.

Hay nói cách khác, giá trị *i* chính là giá trị "vô nghĩa" đối với kiểu số thực trong Python.

Bài tập 2: Thứ tự tính toán trong Python

Là một ngôn ngữ lập trình, các trình biên dịch Python thường xử lý phép toán từ trái sang phải (left-to-right order). Theo đó, các biểu thức logic sẽ bị ảnh hưởng bởi thứ tự tính toán. Ví dụ: trong phép toán AND, nếu yếu tố đầu tiên không thỏa thì ngay lập tức biểu thức sẽ mang giá trị False (sai), nhằm giảm thiểu các tính toán không cần thiết; hoặc với phép toán OR, nếu biểu thức đầu tiên thỏa thì các biểu thức phía sau không cần tính toán.

Sinh viên thực hành các lệnh sau để hiểu được thứ tự tính toán trong Python:

```
>>> a = True
>>> b = False
>>> 1/0

Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    1/0
ZeroDivisionError: integer division or modulo by zero
>>> if (a == b) and (1/0 > 0):
        print ("a=b")
else:
        print ("a khac b")
```

Kết quả câu lệnh if là: ……..

Sinh viên thực hành đoạn lệnh khác:

```
>>> a = True
>>> b = False
>>> if (1/0 > 0) and (a == b):
        print ("a=b")
else:
        print ("a khac b")
```

Sinh viên hãy ghi nhận kết quả đoạn lệnh trên và giải thích.

……..……..……..……..