

THỰC HÀNH TOÁN CAO CẤP

TÀI LIỆU PHỤC VỤ SINH VIÊN NGÀNH KHOA HỌC DỮ LIỆU

Nhóm biên soạn: TS. Hoàng Lê Minh – Khuru Minh Cảnh – Hoàng Thị Kiều Anh – Lê Thị Ngọc Huyền – ...

TP.HCM – Năm 2019

MỤC LỤC

CHƯƠNG 6: HÀM SỐ, DÃY VÀ MỘT SỐ ỨNG DỤNG	3
1. Giới thiệu lập trình đệ quy trong Python	3
2. Dãy số (sequence) đệ quy	4
3. Revisit: Phương pháp Gradient Ascent/Descent.....	11
3.1. Chương trình tổng quát về Gradient Ascent.....	11
3.2. Các lưu ý về giá trị khởi tạo ban đầu (initial value).....	14
3.3. Vai trò của kích thước bước nhảy (step_size) và Epsilon	16
3.4. Lời bàn.....	20
BÀI TẬP CHƯƠNG 6.....	21

CHƯƠNG 6: HÀM SỐ, DÃY VÀ MỘT SỐ ỨNG DỤNG

Mục tiêu:

- Lập trình đệ quy trong Python;
- Dãy số/chuỗi số;
- Revisit phương pháp gradient ascent/descent.

Nội dung chính:

1. Giới thiệu lập trình đệ quy trong Python

Trong toán học, nhiều công thức có thể được thể hiện một phần bằng chính nó. Ví dụ: $n! = 1 \times 2 \times \dots \times n$, góc độ khác, điều này có nghĩa là $n! = n \times (n - 1)!$ Do đó, để tính được $n!$ thì chúng ta phải tính được $(n - 1)!, \dots$ và cuối cùng người ta định nghĩa $0! = 1$. Từ đó, đệ quy là một kỹ thuật lập trình để viết một hàm mà trong hàm lại có lệnh gọi lại chính nó để đáp ứng những định nghĩa toán học.

Thực hành 1: Viết hàm đệ quy tính $n!$

```
>>> import math
```

```
>>> def giaithua(n):
```

```
    if (n==0):
```

```
        return 1
```

```
    else:
```

```
        return n*giaithua(n-1)
```

```
>>> giaithua(3)
```

```
..... ← sinh viên điền kết quả
```

```
>>> giaithua(4)
```

```
..... ← sinh viên điền kết quả
```

Thực hành 2: Viết hàm đệ quy tính dãy số Fibonacci

Dãy số Fibonacci $F(n)$ là dãy số có 2 phần tử ban đầu là 0 và 1, sau đó, các phần tử tiếp theo sẽ là tổng của hai phần tử gần nó nhất. Cụ thể:

$$F(n) = F(n-1) + F(n-2), n \geq 2, F(0) = 0, F(1) = 1$$

Khi đó, chúng ta có thể viết chương trình đệ quy như sau:

```
>>> def fibo(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fibo(n-1) + fibo(n-2)
```

Sinh viên có thể thử nghiệm các giá trị:

```
>>> fibo(4)
.....
>>> fibo(5)
.....
>>> fibo(6)
.....
```

2. Dãy số (sequence) đệ quy

Tổng quát, một dãy số đệ quy là một dãy được cho dưới dạng toán học sau:

$$\begin{cases} a_{n+1} = f(a_n), n = 1, 2, 3, 4, \dots \\ a_1 = \text{Một giá trị cho trước} \end{cases}$$

Giá trị của a_{n+1} được tính theo giá trị a_n . Do đó, dãy số hội tụ khi hiệu $a_{n+1} - a_n$ dần về 0 khi $n \rightarrow \infty$. Điều này suy ra, ít nhất phương trình $f(x) = x$ (hàm f tương ứng với dãy a_n) có nghiệm. Ta gọi các nghiệm đó là điểm bất động (fixed points).

Quy trình tìm giới hạn của các dãy dạng đệ quy như sau:

Bước 1: Giải phương trình tìm nghiệm là các điểm bất động $f(x) = x$.

Bước 2: Nếu dãy số không phải là dãy hằng thì xác định khoảng (a, b) thỏa a và b đều là điểm bất động và vùng có điểm cuối (end point) (có thể là $\pm\infty$) thỏa không tồn tại điểm bất động trong (a, b) và giá trị ban đầu của dãy nằm trong khoảng (a, b) .

Bước 3: Kiểm tra khi nào hàm f sẽ ánh xạ vào trong khoảng (a, b) ở bước 2.

Bước 4: Nếu ở bước 3 kiểm tra f nằm trong (a, b) thì:

Nếu dãy $\{a_n\}$ là đơn điệu thì hoặc $\lim_{n \rightarrow \infty} a_n = b$ hoặc $\lim_{n \rightarrow \infty} a_n = a$.

Nếu f không nằm trong (a, b) thì kiểm tra f nằm trong (a, a_2) , nếu $a < a_2 < a_1$ hoặc kiểm tra f nằm trong (a_1, b) , nếu $a_1 < a_2 < b$.

Sử dụng đệ quy hãy viết chương trình hỗ trợ dự đoán giới hạn của một số dãy số sau:

Thực hành 3: Chứng minh dãy có giới hạn và tìm giới hạn của dãy

$$\begin{cases} a_{n+1} = a_n^{a_n} \\ a_1 = \frac{1}{2} \end{cases}$$

Giải:

Xây dựng chương trình đệ quy:

```
>>> def an_exp_an(n):
    if n == 1:
        return 1.0/2
    else:
        return an_exp_an(n-1)**an_exp_an(n-1)
```

Sinh viên thực hiện một số thử nghiệm:

```
>>> an_exp_an(1)
```

..... ← sinh viên ghi kết quả

>>> an_exp_an(2)

..... ← sinh viên ghi kết quả

>>> an_exp_an(3)

..... ← sinh viên ghi kết quả

>>> an_exp_an(4)

..... ← sinh viên ghi kết quả

>>> an_exp_an(5)

..... ← sinh viên ghi kết quả

>>> an_exp_an(10)

..... ← sinh viên ghi kết quả

Do việc tính toán cần nhiều thời gian nên chúng ta không thể tính toán đến giá trị lớn như $n=50, \dots$. Tuy vậy, chúng ta có thể tận dụng các kết quả trên để nhận định đây là **dãy tăng hay giảm**? ← sinh viên trả lời.

Xét hàm $f(x) = x^x, x > 0$ tương ứng với dãy $a_{n+1} = f(a_n)$

Bước 1: Tìm các điểm bất động với phương trình $f(x) = x$:

$$x^x = x \Leftrightarrow x \ln(x) = \ln(x) \Leftrightarrow \ln(x)(x - 1) = 0 \Leftrightarrow x = 1$$

Bước 2: Nhận định vị trí liên quan đến điểm bất động và vùng biến thiên của hàm f là: $(0,1)$

Bước 3: Xét hàm f trong khoảng $(0,1)$, ta có:

$$0 < f(x) = x^x = e^{x \ln x} < e^0 = 1 \text{ với } 0 < x < 1$$

Lưu ý: do $\ln(x) < 0$ khi $0 < x < 1$ nên $x \ln x < 0$ khi $0 < x < 1$

Bước 4: Từ kết quả trên, chúng ta có thể kết luận dãy $\{a_n\}$ là dãy tăng và nằm trong khoảng $(0,1)$. Do đó, ta có thể kết luận giới hạn của dãy là: $\lim_{n \rightarrow \infty} a_n = 1$

Thực hành 4: Chứng minh dãy có giới hạn và tìm giới hạn của dãy

$$\begin{cases} a_{n+1} = f(a_n) = \frac{5}{6-a_n}, n = 1, 2, 3, \dots \\ a_1 = 4 \end{cases}$$

Giải:

Xây dựng chương trình đệ quy và xét một số giá trị:

```
>>> def bai4(n):
```

```
    if n==1:
```

```
        return 4
```

```
    else:
```

```
        return 5.0/(6-bai4(n-1))
```

```
>>> bai4(5)
```

```
..... ← sinh viên ghi kết quả
```

```
>>> bai4(6)
```

```
..... ← sinh viên ghi kết quả
```

```
>>> bai4(7)
```

```
..... ← sinh viên ghi kết quả
```

```
>>> bai4(10)
```

```
..... ← sinh viên ghi kết quả
```

```
>>> bai4(100)
```

```
..... ← sinh viên ghi kết quả
```

Các kết quả trên để nhận định đây là **dãy tăng hay giảm**? ← sinh viên trả lời.

Xét hàm $f(x) = \frac{5}{6-x}$ tương ứng với dãy $a_{n+1} = f(a_n)$

Bước 1: Tìm các điểm bất động với phương trình $f(x) = x$:

$$\frac{5}{6-x} = x \Leftrightarrow 6x - x^2 = 5 \Leftrightarrow x^2 - 6x + 5 = 0 \Leftrightarrow x = 1 \vee x = 5$$

Bước 2: Nhận định vị trí liên quan đến điểm bất động và vùng biến thiên của hàm f là: (1, 5)

Bước 3: Xét hàm f trong khoảng (1,5), ta có:

$$f'(x) = \frac{5}{(6-x)^2} > 0$$

Nên dễ dàng thấy rằng f là hàm tăng.

Bước 4: Từ kết quả trên, chúng ta có thể kết luận dãy $\{a_n\}$ là dãy đơn điệu và nằm trong khoảng (1,5). Do đó, ta có thể kết luận giới hạn của dãy là: $\lim_{n \rightarrow \infty} a_n = 1$

Thực hành 5: Xét giới hạn của dãy $a_n = \frac{F_{n+1}}{F_n}$ biết dãy Fibonacci được định nghĩa như sau:

$$\begin{cases} F_{n+1} = F_n + F_{n-1} \\ F_0 = 0, F_1 = 1 \end{cases}$$

Với bài này, trước tiên, ta phải thiết lập được “hàm” của dãy, nghĩa là biểu diễn được dạng $a_{n+1} = f(a_n)$. Ta có:

$$a_n = \frac{F_{n+1}}{F_n} = \frac{F_n + F_{n-1}}{F_n} = 1 + \frac{F_{n-1}}{F_n} = 1 + \frac{1}{a_{n-1}}$$

Và lưu ý rằng: $a_1 = \frac{F_2}{F_1} = \frac{F_1 + F_0}{F_1} = 1$

Do vậy, ta xây dựng được dãy như sau:

$$\begin{cases} a_{n+1} = 1 + \frac{1}{a_n}, n = 1, 2, 3, \dots \\ a_1 = 1 \end{cases}$$

Và hàm được xét tương ứng là $g(x) = 1 + \frac{1}{x}$

Chương trình đệ quy để tính các giá trị:

```
>>> def tisoFibo(n):
```

```
    if n==1:
```

```
        return 1
```

```
    else:
```

```
        return 1.0 + 1.0/tisoFibo(n-1)
```



```
>>> for i in range(1, 11):
```

```
    print (i, tisoFibo(i))
```

```
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
```

Nhận xét gì về **10** số hạng đầu tiên? Dãy đơn điệu hay không đơn điệu?.....

Tuy nhiên, chúng ta xét các số hạng lẻ (1,3,5,7,9) và các số hạng chẵn (2,4,6,8,10):

- Dãy số a_1, a_3, a_5, a_7, a_9 có đơn điệu tăng không?.....
- Dãy số $a_2, a_4, a_6, a_8, a_{10}$ có đơn điệu tăng không?.....

Từ đó, ta xét 2 dãy c_n là những $a_{lẻ}$ và d_n là những $a_{chẵn}$, cụ thể như sau:

+ Với dãy c_n :

$$c_1 = a_1$$

$$c_2 = a_3 = g(a_2) = (g \circ g)(c_1)$$

$$c_3 = a_5 = g(a_4) = (g \circ g)(c_2)$$

$$c_4 = a_7 = g(a_6) = (g \circ g)(c_3)$$

Tóm lại:

$$\begin{cases} c_{n+1} = (g \circ g)(c_n) \\ c_1 = a_1 \end{cases}$$

+ Với dãy d_n :

$$d_1 = a_2$$

$$d_2 = a_4 = g(a_3) = (g \circ g)(d_1)$$

$$d_3 = a_6 = g(a_5) = (g \circ g)(d_2)$$

$$d_4 = a_8 = g(a_7) = (g \circ g)(d_3)$$

Tóm lại:

$$\begin{cases} d_{n+1} = (g \circ g)(d_n) \\ d_1 = a_2 \end{cases}$$

Như vậy, 2 dãy $\{c_n\}$ và $\{d_n\}$ đều cùng hàm đệ quy là $g \circ g$ và chỉ khác nhau về giá trị đầu tiên. Thật ra, 2 dãy cũng là phản ánh dãy $\{a_n\}$. Từ đó, chúng ta xét hàm:

$$f(x) = (g \circ g)(x) = 1 + \frac{1}{1 + \frac{1}{x}} = \frac{2x + 1}{x + 1}$$

Dễ thấy, khi đó 2 dãy $\{c_n\}$ và $\{d_n\}$ được thể hiện như sau:

$$\begin{cases} c_{n+1} = f(c_n) = \frac{2c_n + 1}{c_n + 1} \\ c_1 = a_1 = 1 \end{cases}$$

$$\begin{cases} d_{n+1} = f(d_n) = \frac{2d_n + 1}{d_n + 1} \\ d_1 = a_2 = 2 \end{cases}$$

Đến đây, chúng ta thực hiện các bước theo quy trình:

- Bước 1: Tìm điểm bất động của hàm $f(x) = \frac{2x+1}{x+1}$

$$\frac{2x + 1}{x + 1} = x \Leftrightarrow 2x + 1 = x^2 + x \Leftrightarrow x^2 - x - 1 = 0 \Leftrightarrow x = \frac{1 \pm \sqrt{5}}{2}$$

- Bước 2: Xét trên trục số, ta thấy: $-1 < \frac{1-\sqrt{5}}{2} < c_1 < \frac{1+\sqrt{5}}{2} < d_1$

Từ đó, ta xét sự hội tụ của dãy $\{c_n\}$ trong khoảng $(\frac{1-\sqrt{5}}{2}, \frac{1+\sqrt{5}}{2})$ và dãy $\{d_n\}$ trong khoảng $(\frac{1+\sqrt{5}}{2}, +\infty)$

- Bước 3: Khảo sát hàm số f , tính $f' = \frac{1}{(1+x)^2} > 0$ thì ta thấy rằng hàm số f đều tăng ở 2 khoảng $(\frac{1-\sqrt{5}}{2}, \frac{1+\sqrt{5}}{2})$ và $(\frac{1+\sqrt{5}}{2}, +\infty)$, tóm lại là tăng chỉ trừ điểm $x = -1$ không xác định.
- Bước 4: Nhận xét 2 dãy: ta thấy dãy $\{c_n\}$ tăng và dãy $\{d_n\}$ giảm. Từ điều đó, ta có thể kết luận giới hạn của 2 dãy $\{c_n\}$ và $\{d_n\}$ là $\frac{1+\sqrt{5}}{2}$. Tóm lại: $\lim_{n \rightarrow \infty} a_n = \frac{1+\sqrt{5}}{2}$

Yêu cầu sinh viên: Vẽ đồ thị của hàm số $f(x)$ trong khoảng $(\frac{1-\sqrt{5}}{2}, 10)$

>>> ← sinh viên viết lệnh vẽ đồ thị

3. Revisit: Phương pháp Gradient Ascent/Descent

Trong chương 2, chúng ta đã có khái niệm về phương pháp Gradient Ascent. Theo đó, chúng ta biết được phương pháp để tính toán ra các giá trị cực đại cho hàm số. Trong bài này, chúng ta sẽ nghiên cứu rõ hơn phương pháp bằng chương trình tổng quát và giải thích rõ hơn các tham số trong phương pháp:

3.1. Chương trình tổng quát về Gradient Ascent

Chúng ta sẽ sửa đổi chương trình một ít để chương trình thành chương trình tổng quát về tính toán Gradient Ascent.

```

from sympy import Derivative, Symbol, sympify

def grad_ascent(x0, ham_fx, x):
    epsilon = 1e-6
    step_size = 1e-4
    x_old = x0
    x_new = x_old + step_size*ham_fx.subs({x:x_old}).evalf()
    while abs(x_old - x_new) > epsilon:
        x_old = x_new
        x_new = x_old + step_size*ham_fx.subs({x:x_old}).evalf()
    return x_new

if __name__ == '__main__':
    f = input('Nhap ham 1 bien (f): ')
    var = input('Nhap ten bien tuong ung (x): ')
    var0 = float(input('Nhap gia tri khoi dau cho bien x: '))

    try:
        f = sympify(f) # kiem tra ham
    except SympifyError:
        print('Ham nhap khong hop le!')
    else:
        var = Symbol(var)
        d = Derivative(f, var).doit()
        var_max = grad_ascent(var0, d, var)
        print('{0}: {1}'.format(var.name, var_max))
        print('Maximum value: {0}'.format(f.subs({var:var_max})))

```

```
from sympy import Derivative, Symbol, sympify
```

```
def grad_ascent(x0, ham_fx, x):
```

```
    epsilon = 1e-6
```

```
    step_size = 1e-4
```

```
    x_old = x0
```

```
    x_new = x_old + step_size*ham_fx.subs({x:x_old}).evalf()
```

```
    while abs(x_old - x_new) > epsilon:
```

```
        x_old = x_new
```

```
        x_new = x_old + step_size*ham_fx.subs({x:x_old}).evalf()
```

```
return x_new
```

```
if __name__ == '__main__':
```

```
    f = input('Nhập hàm 1 biến (f): ')

```

```
    var = input('Nhập tên biến tương ứng (x): ')

```

```
    var0 = float(input('Nhập giá trị khởi đầu cho biến x: '))

```

```
    try:

```

```
        f = sympify(f) # kiểm tra hàm

```

```
    except SympifyError:

```

```
        print('Hàm nhập không hợp lệ!')

```

```
    else:

```

```
        var = Symbol(var)

```

```
        d = Derivative(f, var).doit()

```

```
        var_max = grad_ascent(var0, d, var)

```

```
        print('{0}: {1}'.format(var.name, var_max))

```

```
        print('Maximum value: {0}'.format(f.subs({var:var_max})))

```

Lưu tập tin và thực hiện thử nghiệm:

- Nhập hàm: $25*25*\sin(2*\theta)/9.8$
- Nhập biến có tên là: θ
- Giá trị khởi đầu là: 0.001

Khi đó, chương trình sẽ tính toán và chạy ra giá trị θ cũng như giá trị cực đại.

```
===== RESTART: D:/gradasc_all.py ===
=====
Nhập ham 1 biến (f): 25*25*sin(2*theta)/9.8
Nhập tên biến tương ứng (x): theta
Nhập giá trị khởi đầu cho biến x: 0.001
theta: 0.785360029379083
Maximum value: 63.7755100185965
```

Các thử nghiệm khác:

- Thử nghiệm 1:

Nhập ham 1 biến (f): cos(t)

Nhập tên biến tương ứng (x): t

Nhập giá trị khởi đầu cho biến x: 0.01

t: ← Sinh viên điền giá trị vào

Maximum value: ← Sinh viên điền giá trị vào

- Thử nghiệm 2:

Nhập ham 1 biến (f): cos(t)+p

Nhập tên biến tương ứng (x): t

Nhập giá trị khởi đầu cho biến x: 0.01

t: ← Sinh viên điền giá trị vào

Maximum value: ← Sinh viên điền giá trị vào

Tuy nhiên, lưu ý rằng hàm **cos(ky)** sẽ không hoạt động đạo hàm bậc nhất của nó là vẫn còn chứa giá trị k trong hàm mà Sympy không biết rõ giá trị k trong đó. Do đó, Sympy không thể tính toán được thuật toán như đoạn code bên trên nếu khi lấy đạo hàm còn hàm bên trong nó. Cụ thể hơn, so sánh $\text{abs}(x_{\text{old}} - x_{\text{new}}) > \text{epsilon}$ không thể thực hiện.

3.2. Các lưu ý về giá trị khởi tạo ban đầu (initial value)

Giá trị khởi tạo của biến để chúng ta bắt đầu lặp thực hiện chương trình đóng vai trò quan trọng trong thuật toán. Xét hàm $x^5 - 30x^3 + 50x$. Chúng ta bắt đầu việc tìm giá trị lớn nhất sử dụng chương trình:

- Giá trị bắt đầu là -2:

Nhap ham 1 bien (f): $x^{**5}-30*x^{**3}+50*x$

Nhap ten bien tuong ung (x): x

Nhap gia tri khoi dau cho bien x: -2

t: ← Sinh viên điền giá trị vào

Maximum value: ← Sinh viên điền giá trị vào

Chương trình dừng khi tìm ra vị trí đỉnh gần nhất (closest peak) nhưng không phải lúc nào cũng là giá trị cực đại toàn cục. Trong trường hợp này, chúng ta chọn giá trị khởi đầu là -2 và nó dừng lại ở đỉnh tương ứng với giá trị cực đại toàn cục (khoảng 706). Xét ví trường hợp tiếp theo với giá trị khởi đầu khác:

- Giá trị bắt đầu là 0.5:

Nhap ham 1 bien (f): $x^{**5}-30*x^{**3}+50*x$

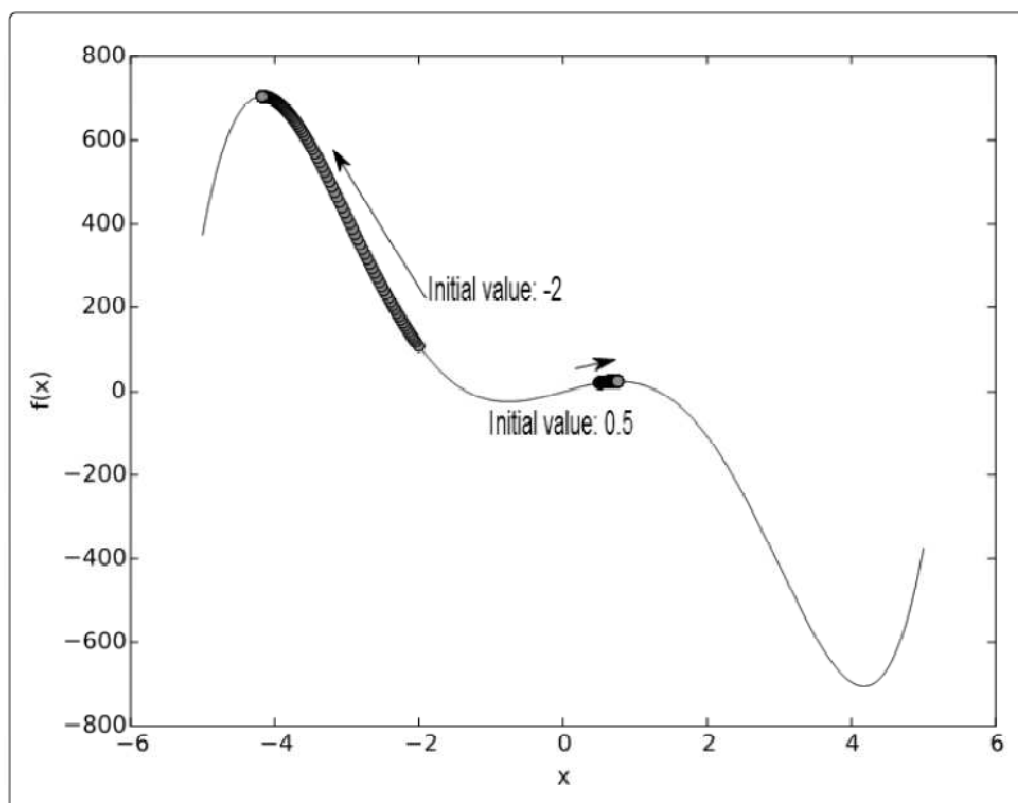
Nhap ten bien tuong ung (x): x

Nhap gia tri khoi dau cho bien x: 0.5

t: ← Sinh viên điền giá trị vào

Maximum value: ← Sinh viên điền giá trị vào

Trong trường hợp này, giá trị cực đại thuật toán gradient ascent tìm được không phải là giá trị lớn nhất của hàm số. Hình dưới đây sẽ cho thấy kết quả của thuật toán gradient ascent cho cả hai trường hợp bên trên:



Do vậy, khi sử dụng phương pháp, giá trị ban đầu cần được lựa chọn kỹ lưỡng. Sau này, một số dạng thay đổi của thuật toán nỗ lực đề cập đến giới hạn này (để cải tiến).

3.3. Vai trò của kích thước bước nhảy (step_size) và Epsilon

Trong thuật toán **gradient ascent**, giá trị θ (hay x) tiếp theo của biến được tính toán bằng phương trình:

$$\theta_{mới} = \theta_{cũ} + \lambda \frac{dR}{d\theta}$$

Trong đó, λ là bước nhảy (step_size). Bước nhảy quyết định khoảng cách của bước tiếp theo. Nên nó phải đủ nhỏ để tránh chuyện vượt qua đỉnh. Do đó, nếu giá trị x gần với giá trị làm cho hàm cực đại và bước nhảy tương đối lớn thì giá trị hàm ở bước tiếp theo có thể sẽ nhỏ cực đại. Và thuật toán sẽ gọi là thất bại! Ngược lại, nếu bước nhảy quá nhỏ thì việc tính toán sẽ nhiều hơn. Trong chương trình tính toán này, chúng ta sử dụng bước nhảy là 10^{-3} , hiển nhiên giá trị này sẽ không phù hợp với mọi hàm.

Giá trị epsilon (ϵ) quyết định khi nào chúng ta nên dừng việc lặp trong thuật toán vì việc thay đổi của x là không đáng kể. Điều này do chúng ta mong đợi đạo hàm cấp 1 của $f(x)$ sẽ triệt tiêu (bằng 0) tại điểm cực đại và lý tưởng hơn là trị tuyệt đối giữa hai giá trị x (hoặc θ) bằng 0, nghĩa là $|x_{mới} - x_{cũ}| = 0$. Tuy nhiên, do có sai số nên hiệu này sẽ không bao giờ bằng 0. Vì vậy, giá trị epsilon được chọn là giá trị gần với 0 để xử lý trường hợp này trong tính toán số thực tế, với sự

ngầm hiểu là x không đổi. Trong chương trình trên, ta sử dụng $\varepsilon = 10^{-6}$ cho tất cả các hàm. Mặc dù giá trị này đủ nhỏ và phù hợp cho hàm có nghiệm $f'(x) = 0$ như $\sin(x)$ nhưng đối với các hàm khác, nếu cần, chúng ta phải điều chỉnh lại giá trị epsilon.

Như vậy, chúng ta có thể cài đặt lại chương trình theo hướng việc tính toán sẽ dừng nếu đạo hàm của hàm số không có nghiệm làm triệt tiêu ($f'(x) = 0$ vô nghiệm). Ví dụ trường hợp e^x hoặc $\log(x)$. Với chương trình mới, nếu nhập các hàm trên thì chương trình sẽ không tiếp tục thực thi. Dưới đây là chương trình được cập nhật:

```
from sympy import Derivative, Symbol, sympify

def grad_ascent(x0, ham_fx, x):

    from sympy import solve, E

    if not solve(ham_fx):

        print('Không thể tiếp tục, phương trình {0}=0 vô nghiệm'.format(ham_fx))

        return

    # như đoạn mã cũ

    epsilon = 1e-6

    step_size = 1e-4

    x_old = x0

    x_new = x_old + step_size*ham_fx.subs({x:x_old}).evalf()

    while abs(x_old - x_new) > epsilon:

        x_old = x_new

        x_new = x_old + step_size*ham_fx.subs({x:x_old}).evalf()

    return x_new

if __name__ == '__main__':

    f = input('Nhập ham 1 biến (f): ')

    var = input('Nhập tên biến tương ứng (x): ')
```

```
var0 = float(input('Nhap gia tri khoi dau cho bien x: '))
```

```
try:
```

```
    f = sympify(f) # kiem tra ham
```

```
except SympifyError:
```

```
    print('Ham nhap khong hop le!')
```

```
else:
```

```
    var = Symbol(var)
```

```
    d = Derivative(f, var).doit()
```

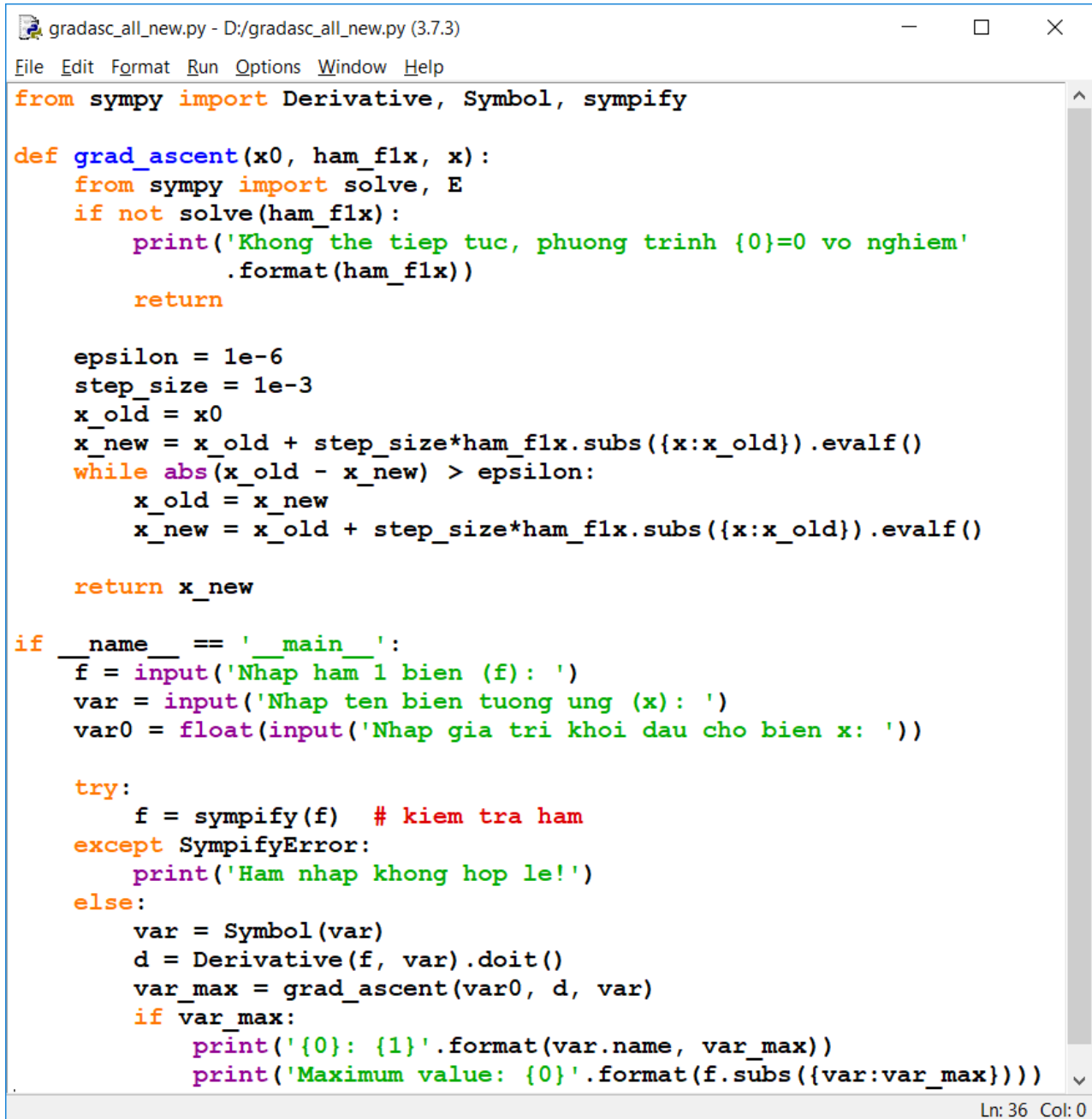
```
    var_max = grad_ascent(var0, d, var)
```

```
    # thêm lệnh if (nằm trong khối lệnh else) để kiểm tra giá trị var_max trước khi kết luận:
```

```
    if var_max:
```

```
        print('{0}: {1}'.format(var.name, var_max))
```

```
        print('Maximum value: {0}'.format(f.subs({var:var_max})))
```



```

gradasc_all_new.py - D:/gradasc_all_new.py (3.7.3)
File Edit Format Run Options Window Help

from sympy import Derivative, Symbol, sympify

def grad_ascent(x0, ham_fx, x):
    from sympy import solve, E
    if not solve(ham_fx):
        print('Khong the tiep tục, phương trình {0}=0 vô nghiệm'
              .format(ham_fx))
        return

    epsilon = 1e-6
    step_size = 1e-3
    x_old = x0
    x_new = x_old + step_size*ham_fx.subs({x:x_old}).evalf()
    while abs(x_old - x_new) > epsilon:
        x_old = x_new
        x_new = x_old + step_size*ham_fx.subs({x:x_old}).evalf()

    return x_new

if __name__ == '__main__':
    f = input('Nhập hàm 1 biến (f): ')
    var = input('Nhập tên biến tương ứng (x): ')
    var0 = float(input('Nhập giá trị khởi đầu cho biến x: '))

    try:
        f = sympify(f) # kiểm tra hàm
    except SympifyError:
        print('Hàm nhập không hợp lệ!')
    else:
        var = Symbol(var)
        d = Derivative(f, var).doit()
        var_max = grad_ascent(var0, d, var)
        if var_max:
            print('{0}: {1}'.format(var.name, var_max))
            print('Maximum value: {0}'.format(f.subs({var:var_max})))

```

Ln: 36 Col: 0

Trong phần chỉnh sửa cho hàm `grad_ascent` này, chúng ta gọi hàm `solve()` của SymPy để kiểm tra sự tồn tại nghiệm của phương trình $f'(x) = 0$. Nếu không có nghiệm, chúng ta in ra thông báo và trả về giá trị rỗng, xem đoạn:

```

print('Khong the tiep tục, phương trình {0}=0 vô nghiệm'.format(ham_fx))

return

```

Từ đó, chúng ta cũng thay đổi hàm __main__. Cụ thể là kiểm tra giá trị trả về của hàm grad_ascent có phải là rỗng hay không; nếu có giá trị (không rỗng) thì chúng ta sẽ in ra; ngược lại, nếu hàm không tồn tại đạo hàm thì chúng ta kết thúc chương trình.

Thực hành: Thử nghiệm với các hàm e^x và $\log x$

Nhap ham 1 bien (f): **log(x)**

Nhap ten bien tuong ung (x): x

Nhap gia tri khoi dau cho bien x: 0.1

..... ← sinh viên điền vào

Nhap ham 1 bien (f): **E**x**

Nhap ten bien tuong ung (x): x

Nhap gia tri khoi dau cho bien x: 0.1

..... ← sinh viên điền vào

3.4. Lời bàn

Thuật toán cùng họ nhưng ngược lại với **gradient ascent** là thuật toán **gradient descent**. Trong thuật toán **gradient descent**, giá trị nhỏ nhất sẽ được tìm kiếm thay vì tìm giá trị lớn nhất (như gradient ascent). Do đó, điểm khác biệt lớn nhất giữa hai hàm là trong khi thuật toán gradient ascent tìm cách “leo lên” thì gradient descent tìm cách “leo xuống”. Sự khác biệt đó chính là sự hình thành giá trị tiếp theo của θ (hay x) như sau:

$$\theta_{mới} = \theta_{cũ} - \lambda \frac{dR}{d\theta}$$

BÀI TẬP CHƯƠNG 6

Bài tập 1: Viết chương trình tính toán 50 phân tử bằng đệ quy và tìm giới hạn các dãy sau:

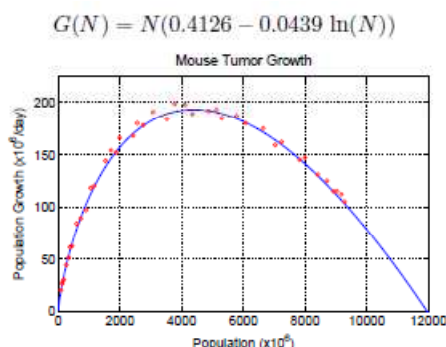
1.
$$\begin{cases} a_{n+1} = \frac{1}{2}(a_n + 8), & n \geq 1 \\ a_1 = 4 \end{cases}$$
2.
$$\begin{cases} a_{n+1} = 2 - \frac{1}{2+a_n}, & n \geq 1 \\ a_1 = 5 \end{cases}$$
3.
$$\begin{cases} a_{n+1} = a_n + \frac{\sin(a_n)}{e^{a_n}}, & n \geq 1 \\ a_1 = \frac{3\pi}{2} \end{cases}$$

Bài tập 2: [Sự phát triển của tế bào ung thư]

Phương trình Gompertz là một trong số những mô hình được sử dụng để mô tả các sự phát triển trong tự nhiên, sinh học. Phương trình có dạng:

$$G(N) = N(b - a \ln(N))$$

Với N là số lượng tế bào/dân số/... ($N > 0$), a và b là những hằng số được tính toán. Ví dụ hình bên dưới là đồ thị của phương trình phát triển của tế bào ung thư trên chuột:



Ở điều kiện cân bằng (equilibrium): $G(N) = N(b - a \ln(N)) = 0$

Do $N > 0$ nên điều kiện cân bằng xảy ra khi: $N_e = e^{b/a}$

Giả định sự phát triển của tế bào ung thư theo phương trình Gompertz cụ thể như sau:

$$G(W) = W(0.5 - 0.05 \ln(W))$$

Với W là trọng lượng (mg). Hãy tìm khối lượng mg khi cân bằng (giải $G(W) = 0$, kết quả là mg) và tìm tỉ lệ tăng nhanh nhất của khối tế bào ung thư (giải $G'(W) = 0$, kết quả là mg/ngày).