**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**VIỆN TOÁN ỨNG DỤNG VÀ TIN HỌC**

**************


# BÁO CÁO MÔN GIẢI TÍCH SỐ

## ĐỀ TÀI:

**"TÌM MIN MAX CỦA HÀM *y= f(X)* TRÊN KHOẢNG ĐÓNG"**

**Giáo viên hướng dẫn : TS. Hà Thị Ngọc Yến**

**Sinh viên thực hiện : Đỗ Diệu Thảo**

**20200599**

***Hà Nội, 10 - 2021***

# MỤC LỤC

# PHẦN MỞ ĐẦU

Giải tích số là môn học về các nghiên cứu, phương pháp về các thuật toán sử dụng sai số xấp xỉ cho các bài toán giải tích. Lớp bài toán đầu tiên được đề cập đến là các phương pháp để giải phương trình phi tuyến f ($x$) = 0. Mà đặc biệt ở đây là tìm giá trị Min-Max của hàm một biến trong khoảng đóng cho trước.

Sau một thời gian tìm hiểu về đề tài "Tìm Min-Max của hàm một biến trên khoảng đóng ", em làm bản báo cáo này để trình bày về nội dung phương pháp, ví dụ, cùng với thuật toán, chương trình cụ thể để giải quyết đề tài này. Trong quá trình làm báo cáo cũng như xây dựng chương trình, thuật toán… không tránh khỏi sai sót, vậy em mong nhận được những lời nhận xét từ cô cũng như mọi người để đề tài này của em hoàn thiện hơn. Em cũng gửi lời cảm ơn cô Hà Thị Ngọc Yến đã giúp em trong quá trình tìm hiểu đề tài này ạ. Hi vọng những phương pháp này sẽ là một công cụ hữu hiệu có thể sử dụng mỗi khi gặp bài toán tìm nghiệm của phương trình, và ứng dụng vào một số bài toán khác!

# PHẦN NỘI DUNG

## A.Bài toán

Tìm giá trị Min, Max của hàm y=$f(x)$ trên [a;b] Giả thiết (a,b) là một khoảng cho trước và $f(x)$ là hàm liên tục trên [a,b].

## B. Phương pháp

Đối với đề tài này em sẽ sử dụng 2 phương pháp khác nhau qua đó phân tích so sánh để làm rõ ưu, nhược điểm của chúng.

### *I. Phương pháp tìm duyệt thông thường*

#### *1. Ý tưởng và xây dựng phương pháp*

Đây là cách tìm min max cơ bản nhất, không cần quan tâm đến dấu của f'($x$) mà chỉ cần f ($x$) liên tục trên [a,b].

Chúng ta sẽ chia đoạn [a;b] thành các đoạn nhỏ và tính giá trị tại các điểm liên tiếp nhau cách nhau một đoạn *step* rất nhỏ và so sánh các giá trị. Sau đó chọn ra điểm làm giá trị f ($x$) nhỏ nhất và lớn nhất.

Sai số là $|x_n - x*|$*(step nhỏ)* . *Step* càng nhỏ thì độ chính xác càng cao nhưng thời gian chạy cũng lâu hơn. Rồi so sánh các f ($x$) và xuất ra min, max

#### *2. Đánh giá phương pháp*

Do phương pháp này duyệt tất cả các giá trị trong [a,b] hơn kém nhau một lượng *step* nhỏ >0, mà không cần quan tâm f'($x$) hay bị ràng buộc bởi các yếu tố khác như hệ số $\eta$ nên chương trình rất tốt, và sai số có thể điều chỉnh theo step dễ dàng ($|x_n - x*| <$*step*).

Tuy nhiên nó cũng có điểm hạn chế thời gian chạy của chương trình khá lâu, các bước lặp lớn.

#### *3. Chương trình*

##### *a. Chương trình*

- **Input:** a, b, f ($x$), *step*
- **Output:** Số lần duyệt *dem*,
  bộ giá trị *xmax* và f (*xmax*)
  *xmin* và f (*xmin*)

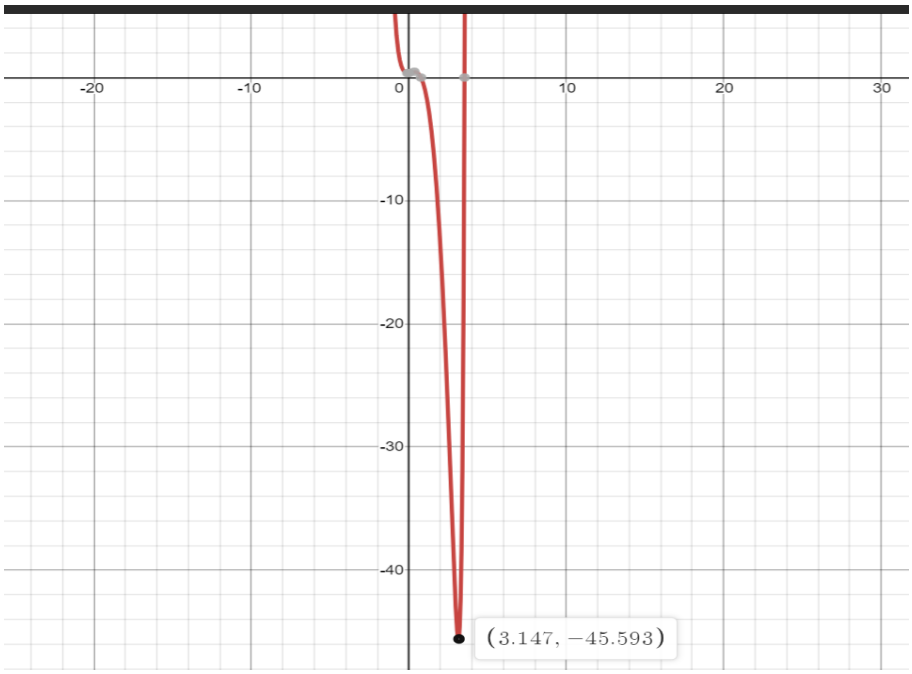
```
#include <bits/stdc++.h>
#define  step 1.0e-4     // Thay đổi các giá trị của step
#define  pi   3.14159265
#define e   2.718281828459
using namespace std;
//-------------------------//
```

```
int dem=0;
double f( double x)
{
    return pow(e,x*x-2*x-1)+x-2*x*x*x;      // Nhập hàm f(x)
}
//-----------------------------------------------//
double x( double a, double b, string s)
{
    double i, xmax, xmin;
    xmax=a; xmin=a; i=a;
    do
    {
        if (f(i) < f(xmin))
            {
                xmin=i;
            }
        if (f(i) > f(xmax))
            {
                xmax=i;
            }
        i+=step;
        dem++;
    }
    while (i<=b);


    if    (s=="max") return xmax;
    else             return xmin;
}
//-----------------------------------------------//
double fx( double a, double b, string s)
{
    double i, fxmax, fxmin;
    fxmax=f(a); fxmin=f(a); i=a;
    do
    {
        if (f(i) < fxmin)
            {
                fxmin=f(i);
            }
        if (f(i) > fxmax)
            {
                fxmax=f(i);
            }
```

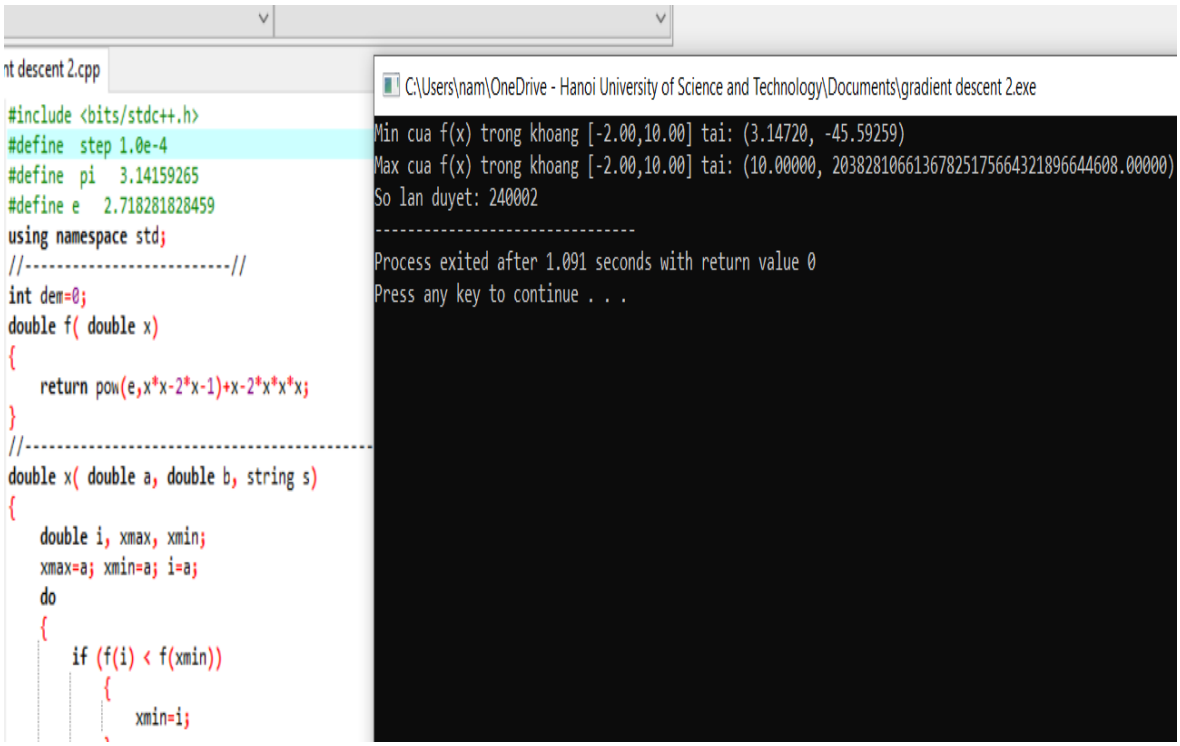
```
        i+=step;
    }
    while (i<=b);


    if    (s=="max") return fxmax;
    else             return fxmin;
}
//-----------------------------------------------//
int main()
{
    double a=-10, b=10;            //Nhập các giá trị a,b
    printf("Min cua f(x) trong khoang [%3.2f,%2.2f] tai: (%2.5f,
%2.5f)\n",a,b,x(a,b,"min"),fx(a,b,"min"));
    printf("Max cua f(x) trong khoang [%3.2f,%2.2f] tai: (%2.5f,
%2.5f)\n",a,b,x(a,b,"max"),fx(a,b,"max"));
    printf("So lan duyet: %d",dem);
}
```

### *b. Ví dụ*

*Ví dụ:* f (x)=$e^{x^2-2x-1} + x - 2x^3$ , trên [-10,10] và trên [-2,10]



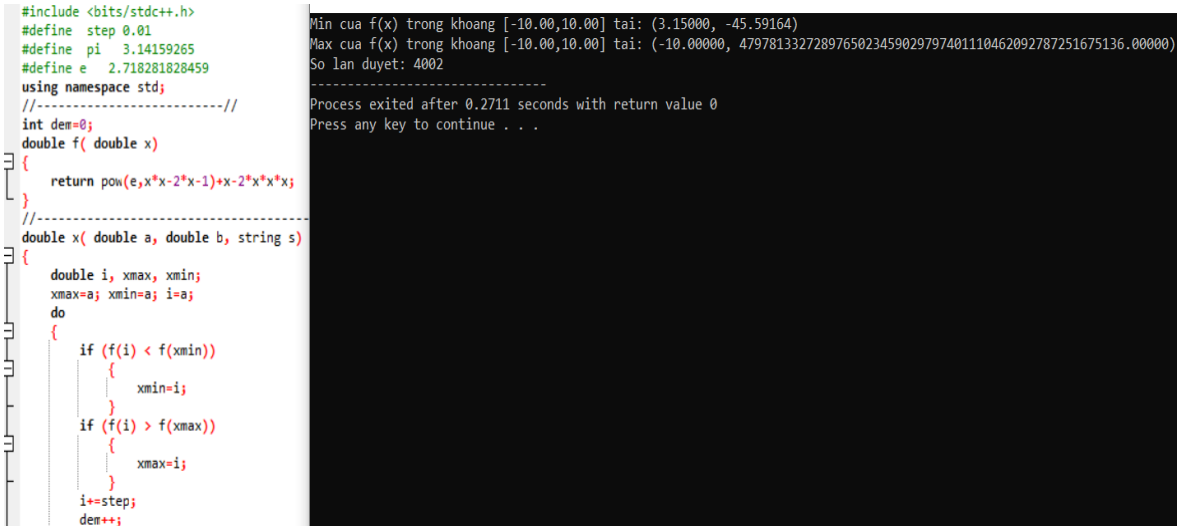
Trên [-2,10] với *step*= $10^{-4}$



```
nt descent 2.cpp

#include <bits/stdc++.h>
#define  step 1.0e-4
#define  pi   3.14159265
#define e   2.718281828459
using namespace std;
//-------------------------//
int dem=0;
double f( double x)
{
    return pow(e,x*x-2*x-1)+x-2*x*x*x;
}
//-----------------------------------------
double x( double a, double b, string s)
{
    double i, xmax, xmin;
    xmax=a; xmin=a; i=a;
    do
    {
        if (f(i) < f(xmin))
            {
                xmin=i;
```

```
C:\Users\nam\OneDrive - Hanoi University of Science and Technology\Documents\gradient descent 2.exe

Min cua f(x) trong khoang [-2.00,10.00] tai: (3.14720, -45.59259)
Max cua f(x) trong khoang [-2.00,10.00] tai: (10.00000, 203828106613678251756643218966446 08.00000)
So lan duyet: 240002
--------------------------------
Process exited after 1.091 seconds with return value 0
Press any key to continue . . .
```

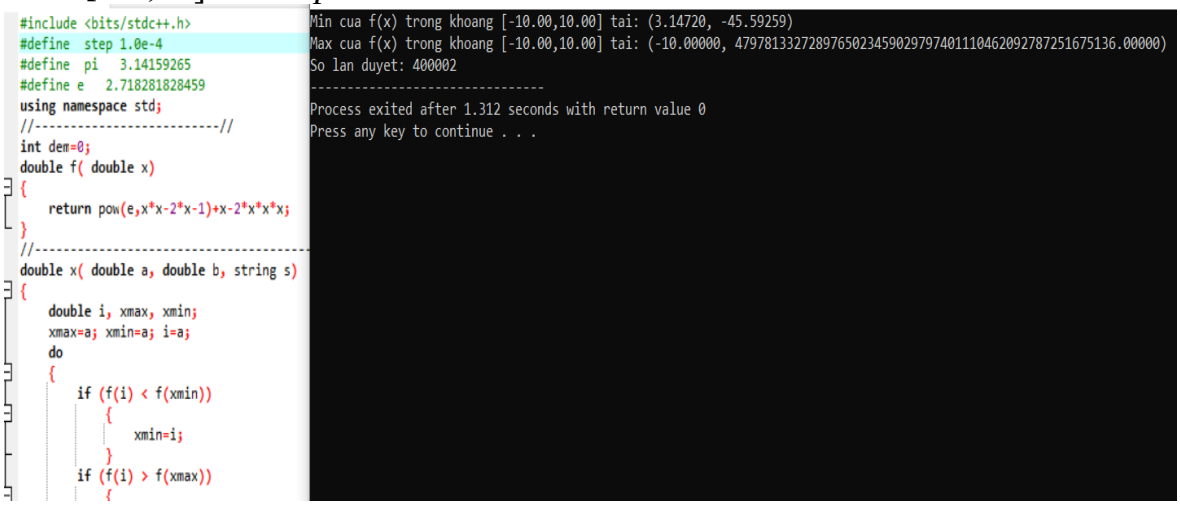
Trên [-10,10] với *step*=0.01



```
#include <bits/stdc++.h>
#define  step 0.01
#define  pi   3.14159265
#define e   2.718281828459
using namespace std;
//-------------------------//
int dem=0;
double f( double x)
{
    return pow(e,x*x-2*x-1)+x-2*x*x*x;
}
//-----------------------------------
double x( double a, double b, string s)
{
    double i, xmax, xmin;
    xmax=a; xmin=a; i=a;
    do
    {
        if (f(i) < f(xmin))
            {
                xmin=i;
            }
        if (f(i) > f(xmax))
            {
                xmax=i;
            }
        i+=step;
        dem++;
```

```
Min cua f(x) trong khoang [-10.00,10.00] tai: (3.15000, -45.59164)
Max cua f(x) trong khoang [-10.00,10.00] tai: (-10.00000, 47978133272897650234590297974011104620927872516751 36.00000)
So lan duyet: 4002
--------------------------------
Process exited after 0.2711 seconds with return value 0
Press any key to continue . . .
```

Trên [-10,10] với *step*=$10^{-4}$



```
#include <bits/stdc++.h>
#define  step 1.0e-4
#define  pi   3.14159265
#define e   2.718281828459
using namespace std;
//-------------------------//
int dem=0;
double f( double x)
{
    return pow(e,x*x-2*x-1)+x-2*x*x*x;
}
//-------------------------------------
double x( double a, double b, string s)
{
    double i, xmax, xmin;
    xmax=a; xmin=a; i=a;
    do
    {
        if (f(i) < f(xmin))
        {
            xmin=i;
        }
        if (f(i) > f(xmax))
        {
```

```
Min cua f(x) trong khoang [-10.00,10.00] tai: (3.14720, -45.59259)
Max cua f(x) trong khoang [-10.00,10.00] tai: (-10.00000, 47978133272897650234590297974011104620927872516751 36.00000)
So lan duyet: 400002
--------------------------------
Process exited after 1.312 seconds with return value 0
Press any key to continue . . .
```

Ta thấy với *step* càng nhỏ thì số lần duyệt càng lớn và độ chính xác càng cao.

## II. Phương pháp Gradient Descent

### *1. Giới thiệu*

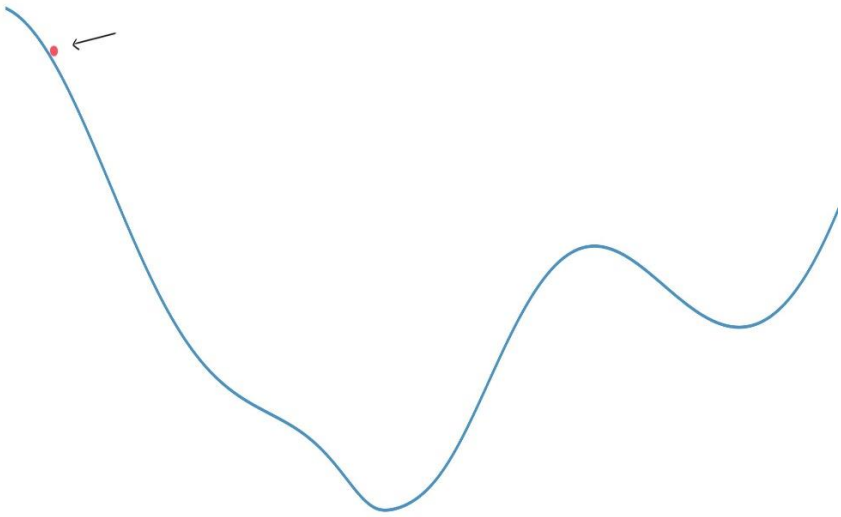
Trong Machine Learning nói riêng và Toán Tối Ưu nói chung, chúng ta thường xuyên phải tìm giá trị nhỏ nhất (hoặc đôi khi là lớn nhất) của một hàm số nào đó. Ví dụ như các hàm mất mát trong hai bài Linear Regression và K-means Clustering. Nhìn chung, việc tìm global minimum của các hàm mất mát trong Machine Learning là rất phức tạp, thậm chí là bất khả thi. Thay vào đó, người ta thường cố gắng tìm các điểm local minimum, và ở một mức độ nào đó, coi đó là nghiệm cần tìm của bài toán.

Các điểm local minimum là nghiệm của phương trình đạo hàm bằng 0. Nếu bằng một cách nào đó có thể tìm được toàn bộ (hữu hạn) các điểm cực tiểu, ta chỉ cần thay từng điểm local minimum đó vào hàm số rồi tìm điểm làm cho hàm có giá trị nhỏ nhất. Tuy nhiên, trong hầu hết các trường hợp, việc giải phương trình đạo hàm bằng 0 là bất khả thi. Nguyên nhân có thể đến từ sự phức tạp của dạng của đạo hàm, từ việc các điểm dữ liệu có số chiều lớn, hoặc từ việc có quá nhiều điểm dữ liệu.

Hướng tiếp cận phổ biến nhất là xuất phát từ một điểm mà chúng ta coi là *gần* với nghiệm của bài toán, sau đó dùng một phép toán lặp để *tiến dần* đến điểm cần tìm, tức đến khi đạo hàm gần với 0. Gradient Descent (viết gọn là GD) và các biến thể của nó là một trong những phương pháp được dùng nhiều nhất.

## 2. *Ý tưởng phương pháp*

Giả sử một người đang lạc ở trên đỉnh núi, trời sắp tối và đỉnh núi rất lạnh khi về đêm nên cần phải đi xuống thung lũng để dựng trại, xung quanh toàn là sương mù nên anh ta không thể xác định chính xác thung lũng ở đâu. Làm thế nào để đi xuống thung lũng?
Vì đam mê toán học, trong đầu anh ấy lóe lên ý tưởng rằng sẽ cố gắng xác định đồ thị hàm số biểu diễn bề mặt của dãy núi, không chần chừ, anh ta bắt tay vào thực hiện.
Bằng một cách nào đó anh ấy đã tìm ra được hình dạng của dãy núi như sau:



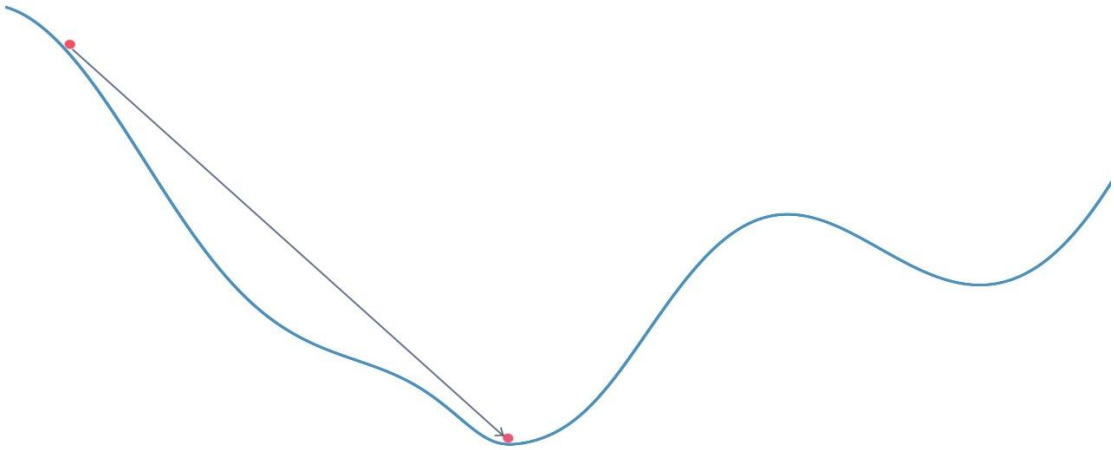
Bằng một phép nội suy, anh ta suy ra được phương trình của đồ thị hàm số trên:

$$f(x) = \frac{\log\left(\,|x|^{(\sin(x)+2)} + 1\right)}{\log(10)}$$

Đạo hàm nó:

$$f'(x) = \frac{(\log(sin(x)+2)+1)(sin(x)+2)^{(sin(x)+2)}}{(|x|^{(sin(x)+2)}+1)log(10)}$$

Giải phương trình đạo hàm bằng 0 và tìm được thung lũng gần đó:



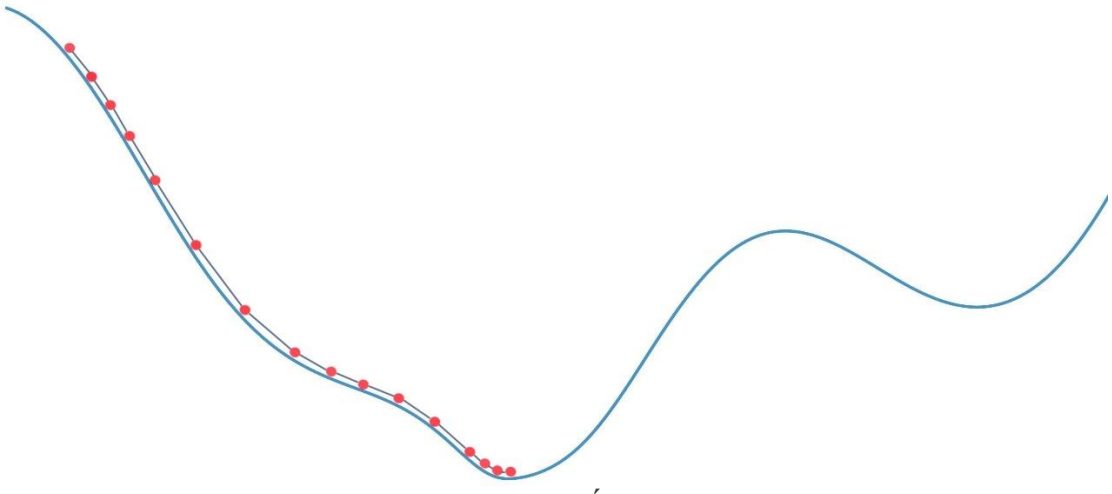
***Anh ta nói xạo đó, dễ gì mà tìm được hàm số, nói gì đến tính toán! Trong thực tế không ai làm như vậy cả.***

Thực chất anh ta làm theo cách bên dưới:

–*Bước 1*: Khảo sát vị trí đang đứng và tất cả những vị trí xung quanh có thể nhìn thấy, sau đó xác định vị trí nào hướng xuống dốc nhiều nhất.

–*Bước2*: Di chuyển đến vị trí đã xác định ở bước 1 .

–*Bước 3*: Lặp lại hai các bước trên đến khi nào tới được thung lũng.



Hay nói cách khác là đi men theo mặt dốc, đây chính là ý tưởng của thuật toán Gradient Descent.

## 3. *Xây dựng công thức*

Đối với cực tiểu

- $x^*$ với $f'(x^*) = 0$
- Tại vị trí $x_n$ bất kì
  Nếu đạo hàm của hàm số tại $x_n$ : $f'(x_n) > 0$ thì $x_n$ nằm về bên phải so với $x^*$ (và ngược lại). Để điểm tiếp theo $x_{n+1}$ gần với $x^*$ hơn, chúng ta cần di chuyển $x_{n+1}$ về phía bên trái, tức về phía âm. Nói cách khác, chúng ta cần di chuyển ngược dấu với đạo hàm:
  $x_{n+1} = x_n + \Delta$. Trong đó $\Delta$ là một đại lượng ngược dấu với $f'(x_n)$
- $x_n$ càng xa $x^*$ về phía bên phải thì $f'(x_n)$ càng lớn hơn 0 (và ngược lại). Vậy, lượng di chuyển $\Delta$ , một cách trực quan nhất, là tỉ lệ thuận với $-f'(x_n)$. Tương tự với cực đại.
- Các nhận xét phía trên cho chúng ta có CTTQ:
  $x_{n+1} = x_n + \text{sign} * \eta * f'(x_n)$ (*)
  (Với sign = -1 khi lặp tìm cực tiểu và sign = 1 khi lặp tìm cực đại)
  Với $\boldsymbol{\eta}$ ($\boldsymbol{eta}$) là một số dương được gọi là learning rate (tốc độ học)
  => $x_n \approx x^*$

***Áp dụng vào bài toán tìm GTLN, GTNN của hàm f (x)=0 trên [a,b]***

Bắt đầu xuất phát từ $x_0$ = a và di chuyển dần về điểm b.

- B1: Nếu $f'(x_0) < 0$ thì điểm cực trị tiếp theo nếu có là cực tiểu. (Tương tự nếu $f'(x_0) > 0$ thì điểm tiếp theo là cực đại).
- B2: Ta sẽ dùng CTTQ (*) với sign = -1( hoặc 1) để lặp $x_0$ đến đủ gần $x^*$. Sau đó ta sẽ tăng $x_0$ thêm 1 khoảng step đủ nhỏ để $x_0$ vượt qua $x^*$ : $x_0 = x_0 + step$. Sau đó quay lại B1 với $x_0$ mới.
- Quá trình lặp lại đến khi $x_0$ = b

Sau khi tìm được các điểm cực đại, cực tiểu chúng ta tiến hành so sánh $f(x^*)$, $f(a)$, $f(b)$ từ đó tìm giác các giá trị lớn nhất (Max) và nhỏ nhất (Min) của hàm đã cho.

## 4. *Điều kiện hội tụ và tốc độ hội tụ*

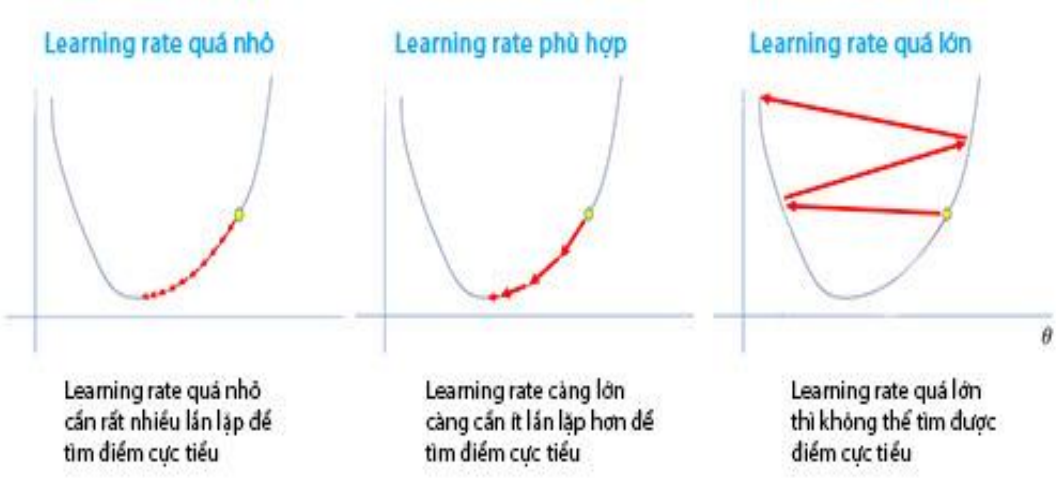
Điều kiện: $f(x)$ liên tục trên [a,b].

Xuất phát từ (*) chúng ta thấy tốc độ hội tụ của GD phụ thuộc vào:

- Điểm khởi tạo ban đầu
- Hệ số learning rate $\eta$

***Tìm η hợp lý***

Như phần trên đã trình bày, tốc độ hội tụ của phương pháp phụ thuộc rất nhiều vào việc lấy $\eta$ ban đầu. Và việc tìm ra $\eta$ hợp lý rất khó.

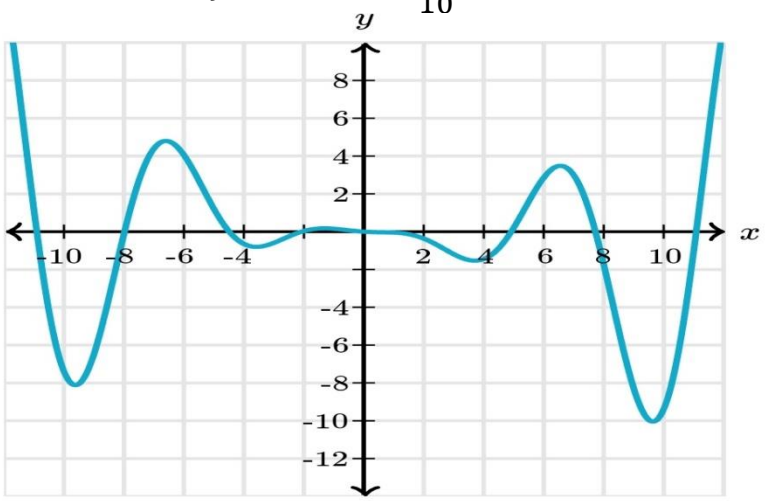



1. Với *learning rate* $\eta$ nhỏ , tốc độ hội tụ rất chậm sẽ ảnh hưởng tới tốc độ của thuật toán rất nhiều, thậm chí không bao giờ tới được đích nếu mình giới hạn số bước lặp.

2. Với *learning rate* $\eta$ lớn, thuật toán có tiến rất nhanh tới *gần đích* sau vài vòng lặp. Tuy nhiên, có trường hợp thuật toán không hội tụ được vì *bước nhảy* quá lớn, khiến nó cứ quẩn quanh ở đích.

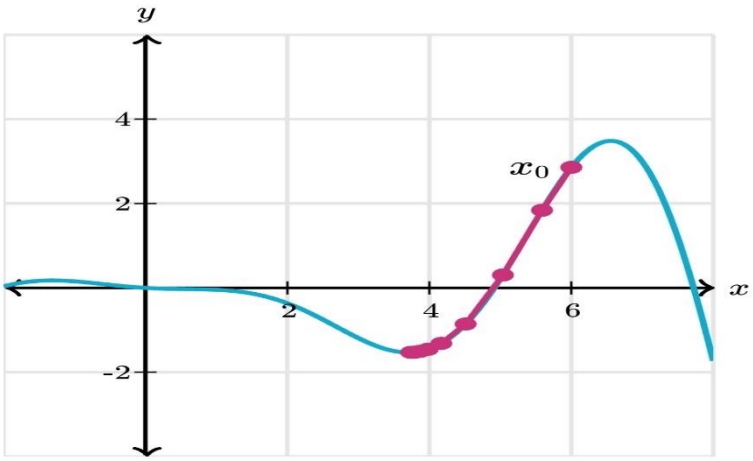
Ta có so sánh rõ hơn ở ví dụ sau:

Xét hàm $f(x) = \frac{x^2 cos(x) - x}{10}$

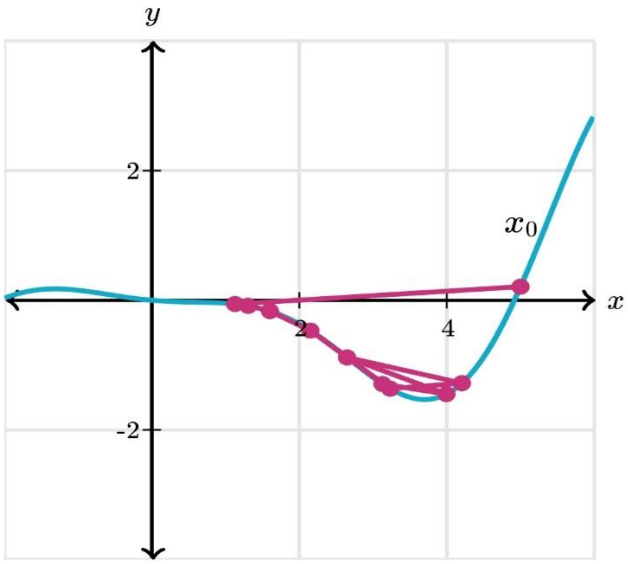



Như chúng ta có thể thấy từ đồ thị, hàm này có nhiều cực tiểu cục bộ. Gradient descent sẽ tìm thấy những cực tiểu khác nhau tùy thuộc vào điểm chọn ban đầu và kích thước bước của chúng.

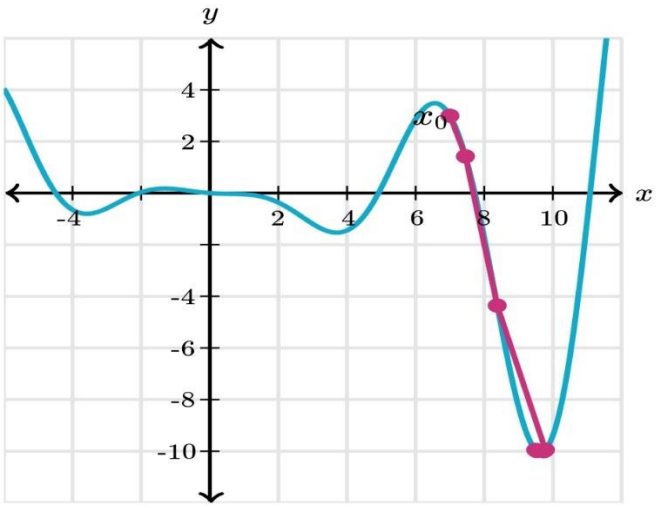
Nếu chúng ta chọn $x_0$=6 và $\eta$ =0.2, gradient desccent di chuyển như hình dưới. Điểm đầu tiên là $x_0$, và các đường nối từng điểm tiếp theo trong chuỗi. Chỉ sau 10 bước, chúng ta đã đến điểm cực tiểu $x$=4.




Nếu chúng ta sử dụng cùng một $x_0$, nhưng $\eta$=1.5, có vẻ như kích thước bước quá lớn để độ dốc gradient có thể hội tụ ở cực tiểu.



Nếu chúng ta bắt đầu lúc $x_0 = 7$, $\eta$=0.2, chúng ta giảm xuống cực tiểu cục bộ hoàn toàn khác.




Như vậy việc lựa chọn learning rate $\eta$ rất quan trọng trong các bài toán thực tế. Việc lựa chọn giá trị này phụ thuộc nhiều vào từng bài toán và phải làm một vài thí nghiệm để chọn ra giá trị tốt nhất. Qua tìm hiểu ta có cách tối ưu hệ số này với 2 trường hợp chính:

Trường hợp 1: Với $\eta$ nhỏ không đủ để bước nhảy đưa $x_1$ vượt qua vị trí của $x^*$ thì 2 điểm liên tiếp sẽ vẫn cùng phía so với $x^*$ , đạo hàm của 2 điểm này sẽ cùng dấu, suy ra $f'(x_1) . f'(x_2) > 0$. Vậy để điều chỉnh ta nên lấy $\eta = \eta * 2$.

Trường hợp 2: Nếu $\eta$ lớn có thể khiến $x_1$ vượt qua vị trí của $x^*$ dẫn đến bỏ sót điểm cực trị cần tìm (có thể ảnh hưởng đến kết quả cuối cùng). Lúc này $f'(x_1) . f'(x_2) < 0$. Để diều chỉnh lấy $\eta = \eta/2$.

***Các điều kiện dừng:**

Điều kiện dừng: là điều kiện để dừng việc cập nhật tham số lại. Nếu không có nó thì không biết khi nào chương trình mới ngừng hết hoạt động. Một chương trình máy tính có 2 kết cục duy nhất là dừng lại được hoặc không bao giờ dừng. Thường người ta sẽ có các cách để thực hiện việc dùng giải thuật sau đây:

- ***Giới hạn số vòng lặp:*** đây là phương pháp phổ biến nhất và cũng để đảm bảo rằng chương trình chạy không quá lâu. Tuy nhiên, một nhược điểm của cách làm này là có thể thuật toán dừng lại trước khi đủ gần với nghiệm.
- ***Kiểm tra giá trị đạo hàm:*** So sánh gradient của nghiệm tại hai lần cập nhật liên tiếp, khi nào giá trị này đủ nhỏ thì dừng lại.

## 5. *Đánh giá phương pháp*

Mặc dù hiện tại phương pháp Gradient Descent được dùng khá phổ biến để tìm Min, Max, nó cũng nhanh hơn và chính xác hơn phương pháp 1 đã nêu nhưng phương pháp này vẫn còn tồn tại những hạn chế nhất định.

***Hạn chế***

• Chỉ làm được với các hàm f ($x$) liên tục trên [a;b]

• Chỉ làm chính xác với các hàm có f ($x$) và f' ($x$) cùng liên tục trên [a;b]

- Do thuật toán này đi tìm các điểm $x^*$ thỏa mãn f' ($x^*$) = 0, nhưng các điểm cực trị thì không nhất thiết có f' ($x^*$) = 0.
- Và f' ($x$) có thể không xác định, thì vẫn thỏa mãn miễn là đổi dấu khi đi qua $x^*$, vì thế nếu f' ($x$) không liên tục thì nó có thể bỏ sót hoặc không chạy được.

• Có thể in ra nhiều giá trị xấp xỉ nhau xung quanh điểm $x^*$

• Đối với các hàm f ($x$) có khoảng cách các cực trị quá bé, nhỏ hơn step cũng không thể chính xác. Bởi:

- Các giá trị $x^*$ chỉ tìm được xấp xỉ chứ không chính xác
- Sau khi tìm được $x^*$ tạm chấp nhận được, ta sẽ tăng i lên 1 đoạn step để nó vượt qua $x^*$, do đó sẽ bị bỏ sót các điểm tới hạn trong khoảng ($x^*$, $x^*$ + *step*)

• Đối với các trường hợp độ lớn của f' ($x$) lớn hơn nhiều so với khoảng cách các cực trị sẽ in thiếu điểm dừng vì độ lớn các bước nhảy tại các vị trí phụ thuộc vào f'($x$) tại đó, gây ra kết quả tìm Min, Max cuối cùng sai.

Chúng ta sẽ tìm hiểu rõ hơn ở phần thuật toán và ví dụ.

## 6. *Thuật toán chương trình*

### a. *Thuật toán*

*Thuật toán chung:*

B1: Chọn $\eta$, *epsilon* đủ bé

Nhập hàm f (x) và giá trị a, b

B2: Bắt đầu xuất phát từ $x_0$ = a và di chuyển dần về điểm b. Sử dụng gói hàm tính và trả về giá trị f'(x)

B3: So sánh f' (x) với 0 và sử dụng CT

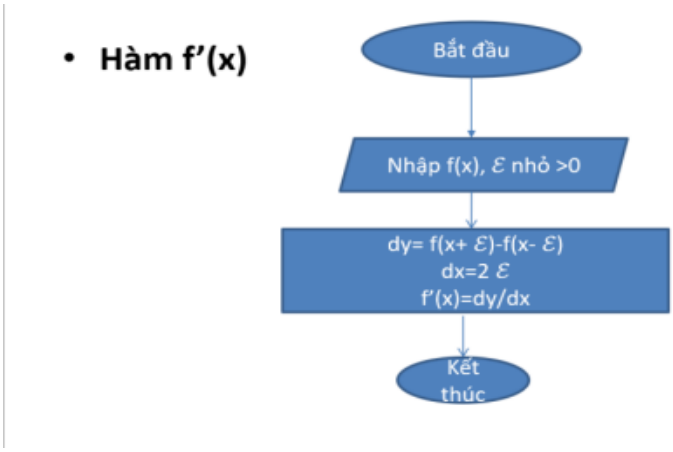
$x = x_0 + \text{sign} * \eta * f'(x_0)$, $x_0 = x$

Vòng lặp dừng khi |f'($x$)|< *epsilon* và trả về các giá trị $x^*$ .Sau đó ta sẽ tăng $x_0$ thêm 1 khoảng step đủ nhỏ để $x_0$ vượt qua $x^*$ :$x_0 = x_0 + step$. Sau đó quay lại B2 với $x_0$ mới.

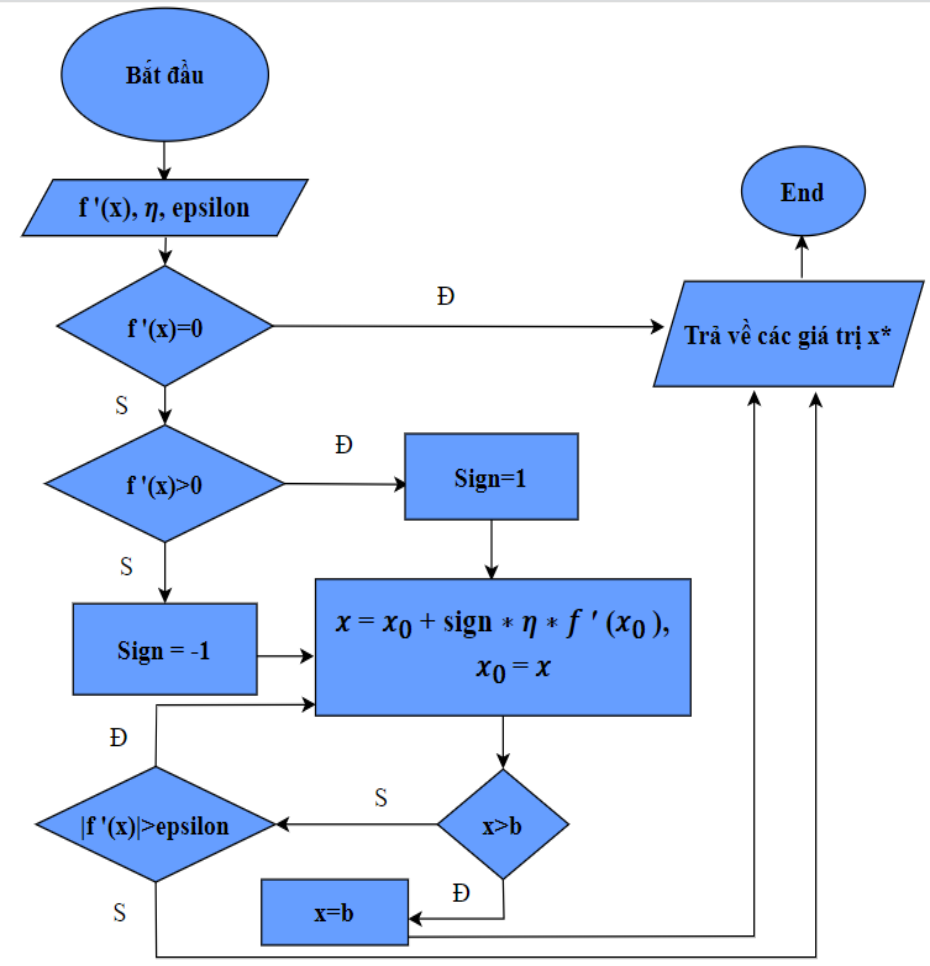
B4: So sánh các giá trị f($x^*$), f(a), f(b) và xuất max,

*Chi tiết các gói*

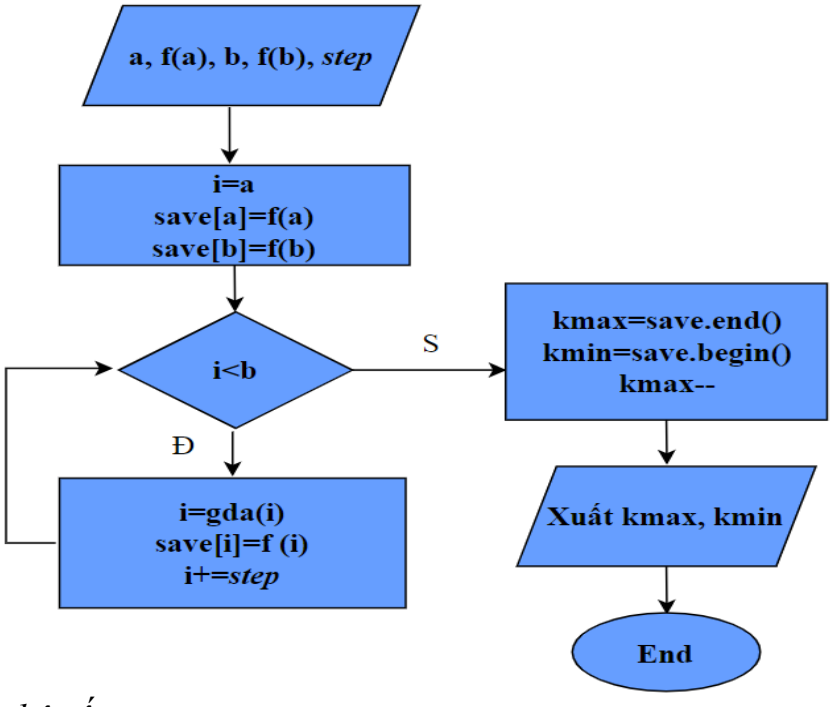
Gói 1: Tính f’(x)




Gói 2: Hàm gda (trả về $x^*$ với f’($x^*$)=0)



Gói 3: Tìm min, max




*(*) Thuật toán tối ưu hệ số eta η*

Input: $x_{init}, \eta_{init}$.
Output: $x^*$
$x_{new} = x_{init} - \eta_{init} * f'(x_{init})$
$flag = f'(x_{init}) * f'(x_{new})$
**while**( $sign(flag) * f'(x_{init}) * f'(x_{new}) > 0$ ):
    $\eta_{init} = \eta_{init} * 2^{sign(flag)}$
    $x_{new} = x_{init} - \eta_{init} * f'(x_{init})$
**end while**

**if**( $sign(flag) = 1$ ):
    $\eta_{best} \longleftarrow \eta_{init}/2$
**else**
    $\eta_{best} \longleftarrow \eta_{init}$
**end if**

%Tiếp theo thay giá trị $x_{new}$ cuối cùng và $\eta_{best}$ vào GD thông thường%

for $i = 1$ to **max_iter**
    $x = x_{new} - \eta_{best} * f'(x_{new})$
    if (( $\| f'(x_{new} \|_2 < \varepsilon$ )):
        break;
    $x_{new} = x$
end for;
$x^* \longleftarrow x$

### *b. Ví dụ chương trình*

- **Input:** a, b, f (*x*), *step*, *eta*, *epsilon*
- **Output:** *xmax* và f (*xmax*)
  *xmin* và f (*xmin*)
  Các *x** và f (*x**), *dem*

*Các hàm sử dụng*

- Hàm f (*x*): Nhập hàm
- Hàm f1 (*x*): Trả về giá trị f'(*x*)
- Hàm gda(x0): Trả về giá trị *x**>*x0* thỏa mãn f'(*x**)=0 ( Các giá trị trả về tăng dần do $x_0$ tăng dần ( i:=a ->b)
- Hàm luutru(): không trả về giá trị. Lưu các giá trị *x**, f'(*x**), a, f (a), b, f (b) vào map
- Hàm xuat1(): không trả về giá trị. Xuất các điểm tới hạn
- Hàm xuat2(): không trả về giá trị. Tìm và xuất max, min của f (*x*)

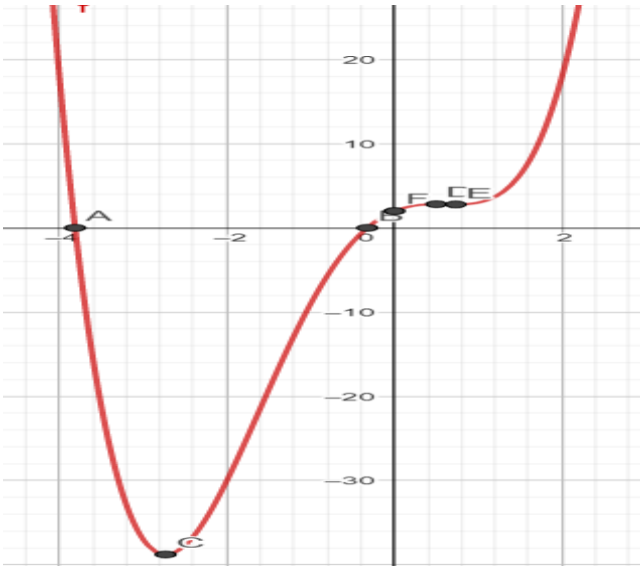
*Chương trình (trường hợp eta tĩnh):*

```
#include <bits/stdc++.h>
#define  eps  1.0e-6
#define  eta  1.0e-4
#define  step 1.0e-3

#define e   2.718281828459
using namespace std;
double  a=-2;
         b=4;
int     sign, dem=0;
map    <double, double> save;
map    <double, double>::iterator k, kmax, kmin;
//-----------------------------------//
double f(double x)  //Nhap ham f(x)
{
    return pow(x,4)+2*pow(x,3)-6*x*x+4*x+2;
}
//-----------------------------------------//
double f1(double x0)  //Ham tra ve f'(x0)
{
       double dy=f(x0+eps)-f(x0-eps),
                 dx=2*eps;
       return dy/dx;
```

```
}
//-------------------------------------------------------------------------
-------------//
double gda(double x0)  //Gradient Desent Asent
{                      //Ham nay tra ve gia tri x*>x0 thoa man f'(x*)=0
                      //Cac gia tri tra ve tang dan vi x0 tang dan (i:=a->b)
    double x=x0;
    if (f1(x0)==0)  return x0;
       if (f1(x0)<0)     sign=-1;
       else              sign= 1;
       while (abs(f1(x0))>eps)
       {
           x=x0+sign*eta*f1(x0);
              x0=x;
              if (x0>b)   return b;
              dem++;
       }
       return x;
}
//-------------------------------------------------------------------------
------//
void luutru()   //Luu cac x* f(x*), a f(a), b f(b) vao map
{
       double i=a;
       save[a]=f(a),
       save[b]=f(b);
    while (i<b)
    {
        i=gda(i);
        save[i]=f(i);
        do
        {
            i+=step;
            if (i>=b) break;
        }
        while ((f1(i)*sign>0) && (abs(f1(i) > f1(i+step))));
    }
}
//-------------------------------------------------//
void xuat1()  //Xuat cac diem toi han
{
       cout<<"Cac diem toi han lan luot la: "<<endl;
       for (k=save.begin(); k!=save.end(); k++)
```

```
        {
            printf("(%5.5f,%5.5f)\n",k->first,k->second);
        }
}
//-------------------------------------------------//
void xuat2() //Tim va xuat max min
{
    kmax=kmin=save.begin();
        for (k=save.begin(); k!=save.end(); k++)
        {
            if (k->second > kmax->second) kmax=k;  // tim f(x) max
            if (k->second < kmin->second) kmin=k; //  tim f(x) min
        }
        printf("Min cua f(x) trong khoang [%5.2f,%5.2f] tai: m
(%5.5f,%5.5f)\n",a,b,kmin->first,kmin->second);
        printf("Max cua f(x) trong khoang [%5.2f,%5.2f] tai: M
(%5.5f,%5.5f)\n",a,b,kmax->first,kmax->second);
        printf("So buoc lap cua GDA la: %d",dem);
}
//----------------------------//
int main()
{
    luutru();
        xuat1();
        xuat2();
}
```

*Ví dụ 1:* f $(x) = x^4+2x^3-6x^2+4x+2$ trên [-4,2] với các eta $\eta$=0.1, $\eta$ =0.01, $\eta$ =$10^{-4}$



Với $\eta = 0.1$

```cpp
#include <bits/stdc++.h>
#include <math.h>
#define  epsilon  1.0e-6
#define  eta  0.1
#define  step 1.0e-3
#define  pi   3.14159265
#define e   2.718281828459
using namespace std;
double  a=-4,
        b=2;
int     sign, dem=0;
map     <double, double> save;
map     <double, double>::iterator k, kmax,
//---------------------------------//
double f(double x)
{
    return pow(x,4)+2*pow(x,3)-6*x*x+4*x+2;
}
//-----------------------------------------
double f1(double x0)
```

```
Cac diem toi han lan luot la:
(-4.00000,18.00000)
(2.00000,18.00000)
Min cua f(x) trong khoang [-4.00, 2.00] tai: m (-4.00000,18.00000)
Max cua f(x) trong khoang [-4.00, 2.00] tai: M (-4.00000,18.00000)
Max cua |f(x)| trong khoang [-4.00, 2.00] la: 18.00000
Min cua |f(x)| trong khoang [-4.00, 2.00] la: 18.00000
So buoc lap cua GDA la: 0
--------------------------------
Process exited after 0.07579 seconds with return value 0
Press any key to continue . . .
```

Kết quả sai vì eta quá lớn, chương trình in ra giá trị của 2 đầu mút.

Với $\eta = 0.01$

```cpp
#include <bits/stdc++.h>
#include <math.h>
#define  epsilon  1.0e-6
#define  eta  0.01
#define  step 1.0e-3
#define  pi   3.14159265
#define e   2.718281828459
using namespace std;
double  a=-4,
        b=2;
int     sign, dem=0;
map     <double, double> save;
map     <double, double>::iterator k, kmax,
//---------------------------------//
double f(double x)
{
    return pow(x,4)+2*pow(x,3)-6*x*x+4*x+2;
}
//-----------------------------------------
```

```
Cac diem toi han lan luot la:
(-4.00000,18.00000)
(-2.73205,-38.78461)
(0.50000,2.81250)
(0.73205,2.78461)
(2.00000,18.00000)
Min cua f(x) trong khoang [-4.00, 2.00] tai: m (-2.73205,-38.78461)
Max cua f(x) trong khoang [-4.00, 2.00] tai: M (-4.00000,18.00000)
Max cua |f(x)| trong khoang [-4.00, 2.00] la: 38.78461
Min cua |f(x)| trong khoang [-4.00, 2.00] la: 0.00000
So buoc lap cua GDA la: 1256
--------------------------------
Process exited after 0.09657 seconds with return value 0
Press any key to continue . . .
```

Kết quả đúng, eta hợp lý.

Với $\eta = 10^{-4}$

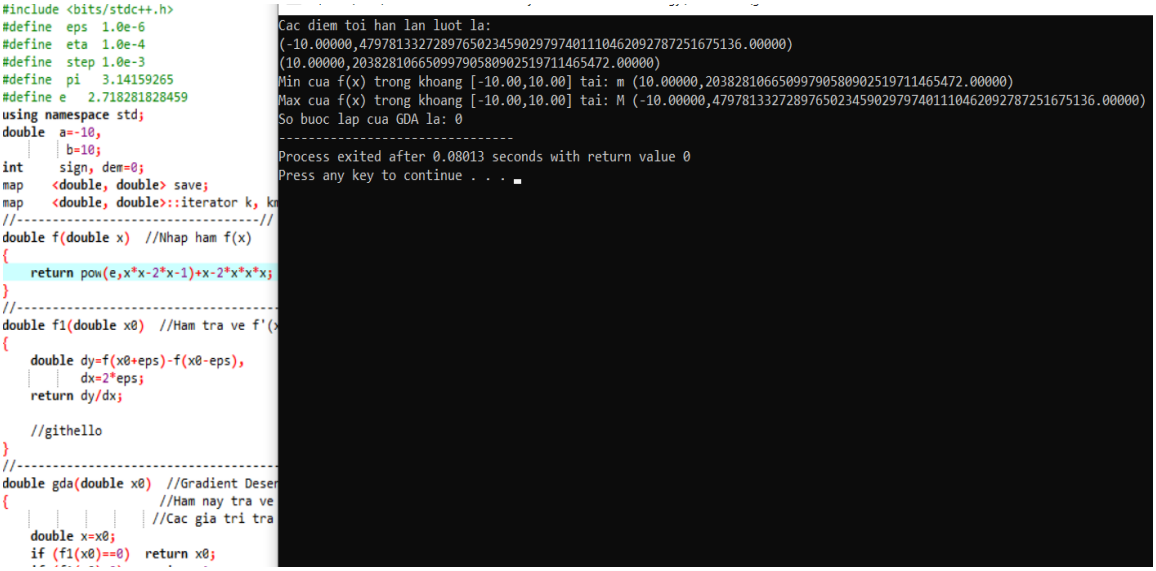
```cpp
#include <bits/stdc++.h>
#include <math.h>
#define  epsilon  1.0e-6
#define  eta  1.0e-4
#define  step 1.0e-3
#define  pi   3.14159265
#define e   2.718281828459
using namespace std;
double  a=-4,
        b=2;
int     sign, dem=0;
map     <double, double> save;
map     <double, double>::iterator k, kmax, kmin;
//---------------------------------//
double f(double x)
{
    return pow(x,4)+2*pow(x,3)-6*x*x+4*x+2;
}
//-----------------------------------------//
double f1(double x0)
```

```
Cac diem toi han lan luot la:
(-4.00000,18.00000)
(-2.73205,-38.78461)
(0.50000,2.81250)
(0.73205,2.78461)
(2.00000,18.00000)
Min cua f(x) trong khoang [-4.00, 2.00] tai: m (-2.73205,-38.78461)
Max cua f(x) trong khoang [-4.00, 2.00] tai: M (-4.00000,18.00000)
Max cua |f(x)| trong khoang [-4.00, 2.00] la: 38.78461
Min cua |f(x)| trong khoang [-4.00, 2.00] la: 0.00000
So buoc lap cua GDA la: 127133
--------------------------------
Process exited after 0.6283 seconds with return value 0
Press any key to continue . . .
```

Kết quả đúng với trường hợp eta nhỏ này, nhưng chạy với số lần lặp lớn hơn và chương trình chạy lâu hơn  so với dùng eta = 0.01.

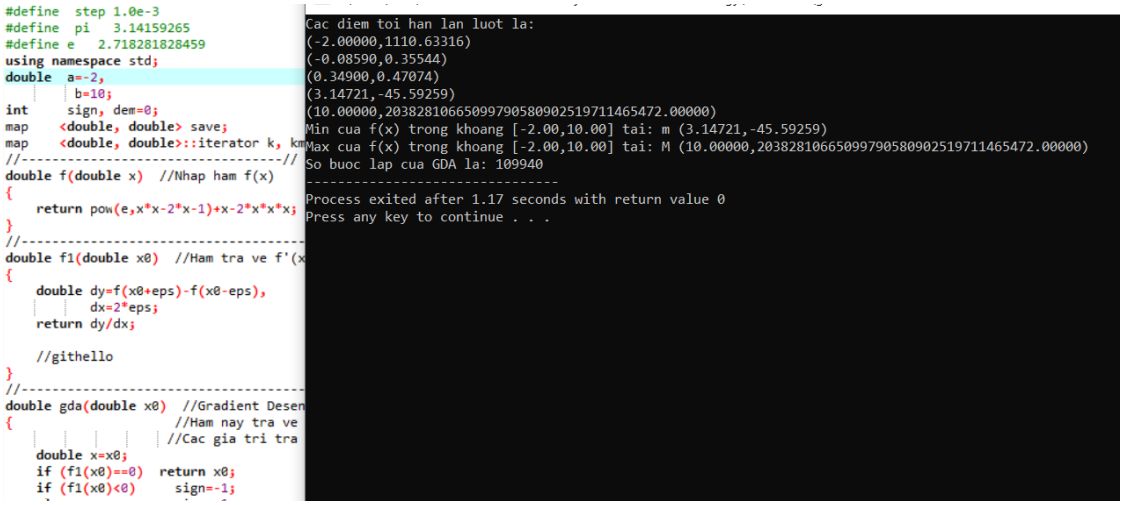
*Kết luận:* Cần điều chỉnh hệ số $\eta$ theo cách hợp lý để ra kết quả chính xác và thời gian chạy tối ưu.

*Ví dụ 2:* f (x)=$e^{x^2-2x-1} + x - 2x^3$, trên [-10,10] và trên [-2,10] với $\eta = 10^{-4}$



```
#include <bits/stdc++.h>
#define  eps  1.0e-6
#define  eta  1.0e-4
#define  step 1.0e-3
#define  pi   3.14159265
#define e   2.718281828459
using namespace std;
double  a=-10,
        b=10;
int     sign, dem=0;
map     <double, double> save;
map     <double, double>::iterator k, km
//---------------------------------//
double f(double x)  //Nhap ham f(x)
{
    return pow(e,x*x-2*x-1)+x-2*x*x*x;
}
//---------------------------------
double f1(double x0)  //Ham tra ve f'(x
{
    double dy=f(x0+eps)-f(x0-eps),
           dx=2*eps;
    return dy/dx;

    //githello
}
//---------------------------------
double gda(double x0)  //Gradient Desen
{                      //Ham nay tra ve
                     //Cac gia tri tra
    double x=x0;
    if (f1(x0)==0)  return x0;
```

```
Cac diem toi han lan luot la:
(-10.00000,479781332728976502345902979740111046209278725167513 6.00000)
(10.00000,203828106650997905809025197114654 72.00000)
Min cua f(x) trong khoang [-10.00,10.00] tai: m (10.00000,203828106650997905809025197114654 72.00000)
Max cua f(x) trong khoang [-10.00,10.00] tai: M (-10.00000,479781332728976502345902979740111046209278725167513 6.00000)
So buoc lap cua GDA la: 0
--------------------------------
Process exited after 0.08013 seconds with return value 0
Press any key to continue . . .
```



```
#define  step 1.0e-3
#define  pi   3.14159265
#define e   2.718281828459
using namespace std;
double  a=-2,
        b=10;
int     sign, dem=0;
map     <double, double> save;
map     <double, double>::iterator k, km
//---------------------------------//
double f(double x)  //Nhap ham f(x)
{
    return pow(e,x*x-2*x-1)+x-2*x*x*x;
}
//---------------------------------
double f1(double x0)  //Ham tra ve f'(x
{
    double dy=f(x0+eps)-f(x0-eps),
           dx=2*eps;
    return dy/dx;

    //githello
}
//---------------------------------
double gda(double x0)  //Gradient Desen
{                      //Ham nay tra ve
                     //Cac gia tri tra
    double x=x0;
    if (f1(x0)==0)  return x0;
    if (f1(x0)<0)     sign=-1;
```

```
Cac diem toi han lan luot la:
(-2.00000,1110.63316)
(-0.08590,0.35544)
(0.34900,0.47074)
(3.14721,-45.59259)
(10.00000,203828106650997905809025197114654 72.00000)
Min cua f(x) trong khoang [-2.00,10.00] tai: m (3.14721,-45.59259)
Max cua f(x) trong khoang [-2.00,10.00] tai: M (10.00000,203828106650997905809025197114654 72.00000)
So buoc lap cua GDA la: 109940
--------------------------------
Process exited after 1.17 seconds with return value 0
Press any key to continue . . .
```

Với hàm f (*x*) này với [a,b]=[-10,10] hàm GDA không thực hiện được vì vậy đã đưa ra kết quả sai.

*Nguyên nhân là do:*

• Trên [-10;10] thì |f'(*x*)| rất lớn.

Ví dụ tại $x_0$ = a = -10: |f'($x_0$)| ≈ $10^{54}$ dẫn đến tại vòng lặp đầu tiên: $x = x_0 - \eta$*f'($x_0$) > b và vòng lặp kết thúc (bước nhảy quá lớn, hệ số $\eta$ không đủ nhỏ để hãm f'($x_0$) lại). Lúc đó sẽ không có điểm dừng và Min Max chính là giá trị tại 2 đầu mút [-10;10].

Với trường hợp này thì đã có phương pháp thứ nhất xử lý, phương pháp thứ nhất đã đưa ra kết quả tương đối chính xác với sai số *step* .

Với [a,b]=[-2,10] chương trình chạy ra kết quả đúng như hình trên, $\eta$ đủ nhỏ để làm cho bước nhảy không quá lớn. Ta thấy thời gian chạy của chương trình nhanh hơn nhiều so với phương pháp duyệt thông thường.

## 7. Ứng dụng

Với Gradient Descent, phương pháp này ứng dụng rất nhiều trong **Machine Learning, Deep Learning** để tìm điểm cực đại, cực tiểu của hàm một biến, và đặc biệt với hàm nhiều biến hay đi giải phương trình f'(x)=0 bằng các biến thể của Gradient decent như: **Newton's method, Mini-batch Gradient Descent, Batch Gradient Descent, Stochastic Gradient Descent**.