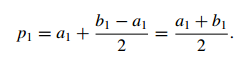
**Bài 3.1 Phương pháp chia đôi (The Bisection Methods)**

BisectionTechnique

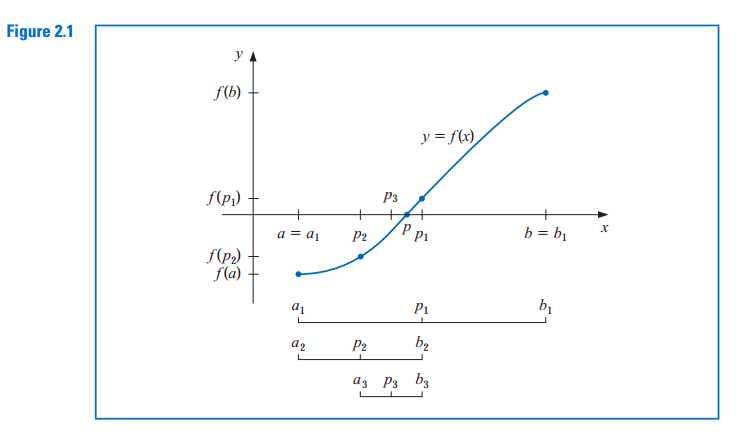
Suppose *f* is a continuous function defined on the interval [*a*, *b*], with *f (a)* and *f (b)* of opposite sign. The Intermediate Value Theorem implies that a number *p* exists in *(a*, *b)* with *f ( p)* = 0. Although the procedure will work when there is more than one root in the interval *(a*, *b)*, we assume for simplicity that the root in this interval is unique. The method calls for a repeated halving (or bisecting) of subintervals of [*a*, *b*] and, at each step, locating the half containing *p*.

Giả sử f là hàm liên tục được xác định trên khoảng [a, b], với f (a) và f (b) của dấu trái. Định lý giá trị trung gian ngụ ý rằng một số p tồn tại trong (a, b) với f (p) = 0. Mặc dù quy trình sẽ hoạt động khi có nhiều hơn một gốc trong khoảng (a, b), chúng ta giả định rằng gốc trong khoảng thời gian này là duy nhất. Phương thức này yêu cầu một nửa phân đoạn lặp lại (hoặc chia đôi) của các khoảng con [a, b] và, ở mỗi bước, định vị nửa chứa p.

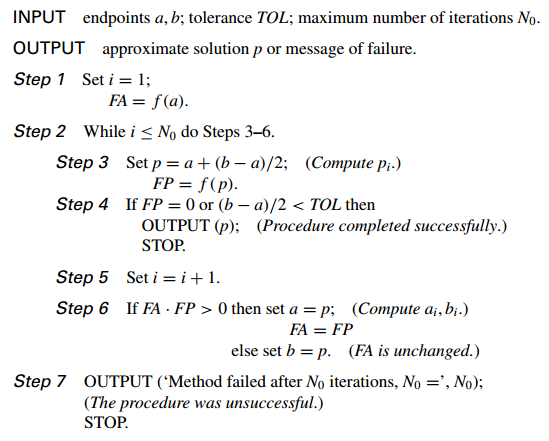
To begin, set *a*1 = *a* and *b*1 = *b*, and let *p*1 be the midpoint of [*a*, *b*]; that is,

**

• If *f ( p*1*)* = 0, then *p* = *p*1, and we are done.  
• If *f ( p*1*)* ≠ 0, then *f ( p*1*)* has the same sign as either *f (a*1*)* or *f (b*1*)*.  
• If *f ( p*1*)* and *f (a*1*)* have the same sign, *p* ∈ *( p*1, *b*1*)*. Set *a*2 = *p*1 and *b*2 = *b*1.  
• If *f ( p*1*)* and *f (a*1*)* have opposite signs, *p* ∈ *(a*1, *p*1*)*. Set *a*2 = *a*1 and *b*2 = *p*1.  
Then reapply the process to the interval [*a*2, *b*2]. This produces the method described in Algorithm 2.1. (See Figure 2.1.

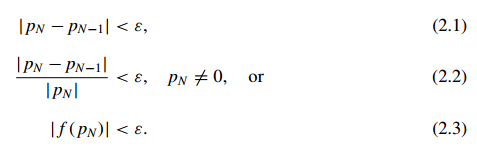


**Bisection**To find a solution to *f (x)* = 0 given the continuous function *f* on the interval [*a*, *b*], where *f (a)* and *f (b)* have opposite signs:



Other stopping procedures can be applied in Step 4 of Algorithm 2.1 or in any of the iterative techniques in this chapter. For example, we can select a tolerance *ε >* 0 and generate *p*1, *. . .* , *pN* until one of the following conditions is met:

Các thủ tục dừng khác có thể được áp dụng trong Bước 4 của Thuật toán 2.1 hoặc trong bất kỳ các kỹ thuật lặp lại trong chương này. Ví dụ, chúng ta có thể chọn một dung sai ε> 0 và tạo p1,. . . , pN cho đến khi đáp ứng một trong các điều kiện sau:



Unfortunately, difficulties can arise using any of these stopping criteria. For example,  
there are sequences { *pn*}∞ *n*=0 with the property that the differences *pn* - *pn*-1 converge to  
zero while the sequence itself diverges. (See Exercise 17.) It is also possible for *f ( pn)* to  
be close to zero while *pn* differs significantly from *p*. (See Exercise 16.) Without additional  
knowledge about *f* or *p*, Inequality (2.2) is the best stopping criterion to apply because it  
comes closest to testing relative error.  
When using a computer to generate approximations, it is good practice to set an upper  
bound on the number of iterations. This eliminates the possibility of entering an infinite  
loop, a situation that can arise when the sequence diverges (and also when the program is  
incorrectly coded). This was done in Step 2 of Algorithm 2.1 where the bound *N*0 was set  
and the procedure terminated if *i > N*0.  
Note that to start the Bisection Algorithm, an interval [*a*, *b*] must be found with *f (a)* ·  
*f (b) <* 0. At each step the length of the interval known to contain a zero of *f* is reduced  
by a factor of 2; hence it is advantageous to choose the initial interval [*a*, *b*] as small as  
possible. For example, if *f (x)* = 2*x*3 - *x*2 + *x* - 1, we have both

Thật không may, khó khăn có thể phát sinh bằng cách sử dụng bất kỳ tiêu chí dừng nào. Ví dụ,

có các chuỗi {pn} ∞ n = 0 với thuộc tính mà sự khác biệt pn - pn -1 hội tụ về 0 trong khi trình tự phân kỳ. (Xem Bài tập 17.) Cũng có thể cho f (pn) gần bằng không trong khi pn khác đáng kể so với p. (Xem Bài tập 16.) Nếu không có kiến ​​thức bổ sung về f hoặc p, bất bình đẳng (2.2) là tiêu chuẩn dừng tốt nhất để áp dụng vì nó đến gần nhất để kiểm tra lỗi tương đối.

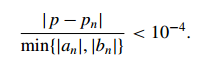
Khi sử dụng một máy tính để tạo ra xấp xỉ, nó là thực hành tốt để thiết lập một giới hạn trên về số lần lặp lại. Điều này giúp loại bỏ khả năng nhập một vòng lặp vô hạn, một tình huống có thể phát sinh khi trình tự phân kỳ (và cả khi chương trình được mã hóa sai). Điều này đã được thực hiện trong Bước 2 của Thuật toán 2.1 trong đó N0 bị ràng buộc được thiết lập và thủ tục chấm dứt nếu i> N0.

Lưu ý rằng để bắt đầu thuật toán bisection, một khoảng [a, b] phải được tìm thấy với f (a) ·

f (b) <0. Ở mỗi bước, độ dài của khoảng được biết là chứa số không f được giảm theo hệ số 2; do đó rất thuận lợi để chọn khoảng thời gian ban đầu [a, b] càng nhỏ càng tốt. Ví dụ: nếu f (x) = 2x3 - x2 + x - 1, chúng tôi có cả hai



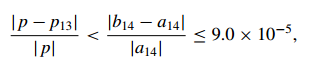
so the Bisection Algorithm could be used on [-4, 4] or on [0, 1]. Starting the Bisection  
Algorithm on [0, 1] instead of [-4, 4] will reduce by 3 the number of iterations required to  
achieve a specified accuracy.  
The following example illustrates the Bisection Algorithm. The iteration in this example  
is terminated when a bound for the relative error is less than 0.0001. This is ensured by  
having

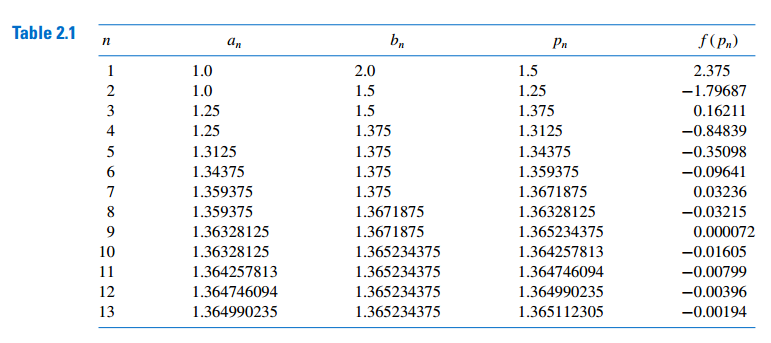


**Example 1** Show that *f (x)* = *x*3 + 4*x*2 - 10 = 0 has a root in [1, 2], and use the Bisection method to  
determine an approximation to the root that is accurate to at least within 10-4.  
***Solution*** Because *f (*1*)* = -5 and *f (*2*)* = 14 the Intermediate Value Theorem 1.11 ensures  
that this continuous function has a root in [1, 2].  
For the first iteration of the Bisection method we use the fact that at the midpoint of  
[1, 2] we have *f (*1.5*)* = 2.375 *>* 0. This indicates that we should select the interval [1, 1.5]  
for our second iteration. Then we find that *f (*1.25*)* = -1.796875 so our new interval  
becomes [1.25, 1.5], whose midpoint is 1.375. Continuing in this manner gives the values  
in Table 2.1. After 13 iterations, *p*13 = 1.365112305 approximates the root *p* with an error



Since |*a*14| *<* | *p*|, we have





so the approximation is correct to at least within 10-4. The correct value of *p* to nine decimal  
places is *p* = 1.365230013. Note that *p*9 is closer to *p* than is the final approximation *p*13.  
You might suspect this is true because |*f ( p*9*)*| *<* |*f ( p*13*)*|, but we cannot be sure of this  
unless the true answer is known.  
The Bisection method, though conceptually clear, has significant drawbacks. It is relatively slow to converge (that is, *N* may become quite large before | *p* - *pN*| is sufficiently small), and a good intermediate approximation might be inadvertently discarded. However, the method has the important property that it always converges to a solution, and for that  
reason it is often used as a starter for the more efficient methods we will see later in this chapter.

***Theorem 2.1*** Suppose that *f* ∈ *C*[*a*, *b*] and *f (a)*·*f (b) <* 0. The Bisection method generates a sequence  
{ *pn*}∞ *n*=1 approximating a zero *p* of *f* with



***Proof*** For each *n* ≥ 1, we have



Since



for all *n* ≥ 1, it follows that



the sequence { *pn*}∞ *n*=1 converges to *p* with rate of convergence *O*  21 *n* ; that is,



It is important to realize that Theorem 2.1 gives only a bound for approximation error  
and that this bound might be quite conservative. For example, this bound applied to the  
problem in Example 1 ensures only that



but the actual error is much smaller:



**Example 2** Determine the number of iterations necessary to solve *f (x)* = *x*3 + 4*x*2 - 10 = 0 with  
accuracy 10-3 using *a*1 = 1 and *b*1 = 2.  
***Solution*** We we will use logarithms to find an integer *N* that satisfies



Logarithms to any base would suffice, but we will use base-10 logarithms because the tolerance is given as a power of 10. Since



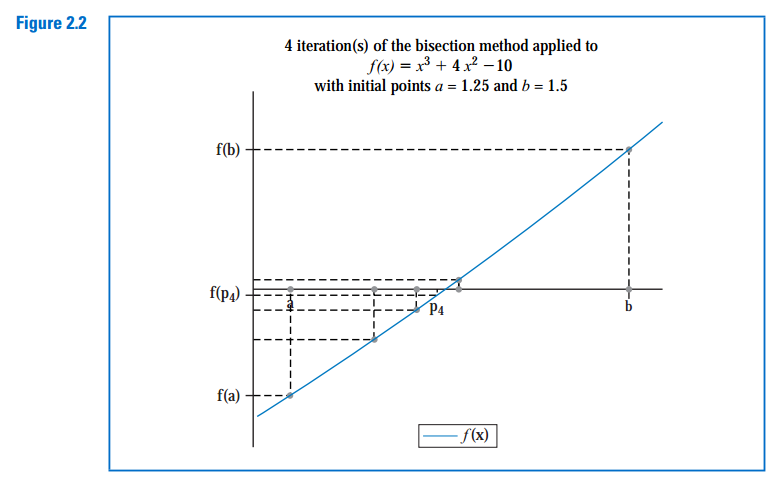
implies that



we have



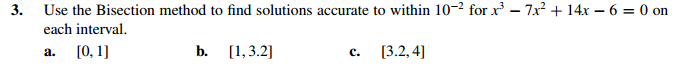
Hence, ten iterations will ensure an approximation accurate to within 10-3.  
Table 2.1 shows that the value of *p*9 = 1.365234375 is accurate to within 10-4. Again,  
it is important to keep in mind that the error analysis gives only a bound for the number of  
iterations. In many cases this bound is much larger than the actual number required.  
Maple has a *NumericalAnalysis* package that implements many of the techniques we  
will discuss, and the presentation and examples in the package are closely aligned with this  
text. The Bisection method in this package has a number of options, some of which we will  
now consider. In what follows, Maple code is given in *black italic* type and Maple response  
in cyan.  
Load the *NumericalAnalysis* package with the command  
*with*(*Student*[*NumericalAnalysis*])  
which gives access to the procedures in the package. Define the function with

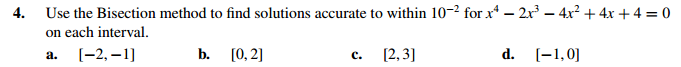


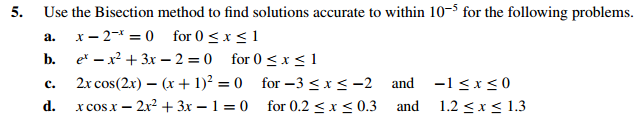
Bài tập Phương pháp chia đôi

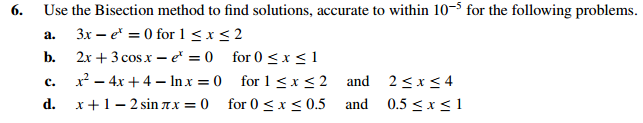




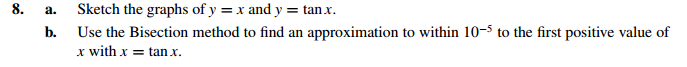


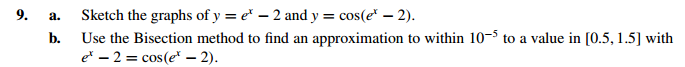


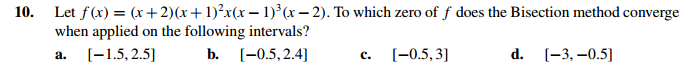
















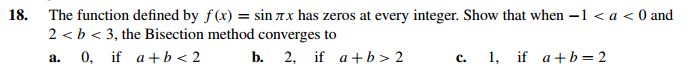


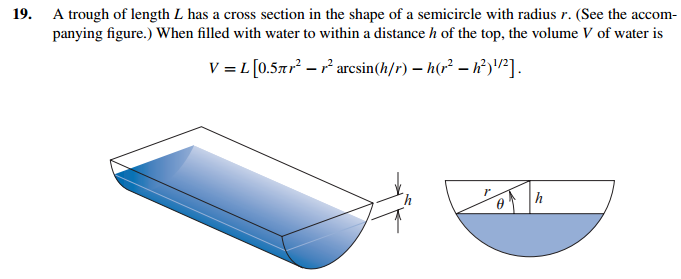




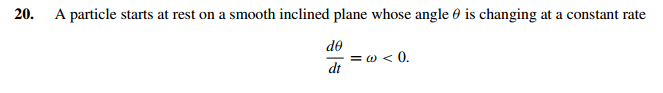


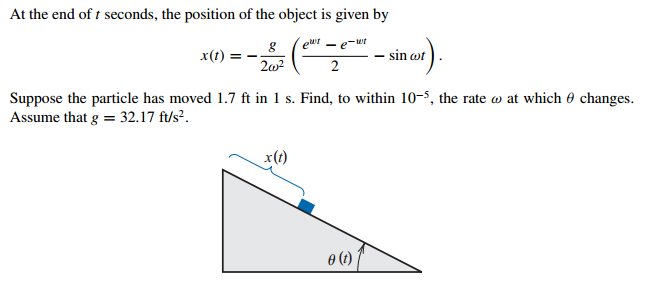












The Latin word signum means “token” or “sign”. So the signum function quite naturally returns the sign of a number (unless the number is 0)

Chữ signum Latin có nghĩa là “token” hoặc “sign”. Vì vậy, chức năng signum khá tự nhiên trả về dấu của một số (trừ khi con số là 0)