TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN TOÁN ỨNG DỤNG VÀ TIN HỌC

BÁO CÁO MÔN HỌC

GIẢI TÍCH SỐ

ĐỀ TÀI 4

PHƯƠNG PHÁP TIẾP TUYẾN GIẢI PHƯƠNG TRÌNH PHI TUYẾN

GV hướng dẫn: TS. HÀ THỊ NGỌC YẾN

Nhóm sinh viên thực hiện: Nhóm 12 lớp 116439

Họ và tên	MSSV
Nguyễn Quang Huy	20185454
Đỗ Thị Thanh Châu	20185434
Nguyễn Đức Hoàng	20185360
Nhữ Thị Lan	20185462
Lê Thành Trung	20185486

Hà Nội, 2020

Mở đầu

Giải tích số (Numerial Analysis) là một môn khoa học nghiên cứu cách giải gần đúng, chủ yếu là giải số, các phương trình, các bài toán xấp xỉ hàm số và các bài toán tối ưu.

Ban đầu, toán học phát sinh do nhu cầu giải các bài toán thực tế như tính diện tích đất đai, quỹ đạo của các vì sao, đường đi của các con tàu buôn trên biển,... nên thuật ngữ Toán học đồng nghĩa với Toán học tính toán. Cùng với sự phát triển của toán học và các ngành khoa học khác, toán học được chia thành toán lý thuyết và toán ứng dụng. Tất cả những nhà toán học vĩ đại như Newton, Euler, Lagrange, Gauss,... Đều có những công trình nền móng trong Giải tích số.

Trong những năm 50 trở lại đây, đặc biệt là những năm 80, Giải tích sô đặc biệt phát triển cùng với sự phát triển của Tin học. Nếu toán lý thuyết chỉ quan tâm đến việc chứng minh tồn tại nghiệm, khảo sát dáng điệu và một số tính chất định tính của nghiệm thì toán tính đề xuất các thuật toán giải trên máy. Giải tích số đặc biệt quan tâm đến các vấn đề: thời gian máy, bộ nhớ cần sử dụng, tốc độ hội tụ và sự ổn định của thuật toán.

Xấp xỉ hàm số, giải gần đúng các phương trình và giải gần đúng các bài toán tối ưu là 3 nhiệm vụ chính vô cùng quan trọng của Giải tích số. Trong bài báo cáo này, chúng em sẽ nghiên cứu một phương pháp được sử dụng để giải gần đúng nghiệm của phương trình $f(x) = 0$, đó là phương pháp tiếp tuyến. Bài báo cáo chuẩn bị trong thời gian ngắn, cũng là lần đầu làm, nên chúng em không tránh khỏi có những sai sót, rất mong được cô và các bạn góp ý.

Chúng em xin cảm ơn cô Hà Thị Ngọc Yến đã dạy lớp chúng em và hướng dẫn chúng em hoàn thành báo cáo này.

Hà Nội, ngày 9 tháng 6 năm 2020

Nhóm 12 lớp 116439

Mục lục

Danh sách hình vẽ

Listings

Chỉnh sửa, bổ sung so với báo cáo cũ

- Sử dụng format mới trình bày các định lý và các ví dụ
- Sửa lỗi chính tả và diễn đạt
- Sửa kí hiệu sai trong sơ đồ khối thuật toán *Newton*
- Sửa mũi tên sai trong sơ đồ khối thuật toán *Newt - Safe*
- Đặt vấn đề theo cách khác cho bài toán giải phương trình $\ln x - 1 = 0$
- Chụp lại các ảnh màn hình Console to rõ hơn
- Bổ sung phép chứng minh sơ lược thuật toán "tựa" *Newton* xấp xỉ đạo hàm bởi $f'(x_0)$ trong phần tính đạo hàm ở ví dụ 7
- Tách code thành các phần nhỏ bổ sung vào các ý phân tích để minh họa rõ hơn trong các ví dụ 7, 19, 20
- Bổ sung minh chứng (ảnh màn hình CASIO, Geobegra) phần kiểm tra tính đúng đắn của chương trình ở ví dụ 7
- Sửa điều kiện dừng sai trong code thuật toán chia đôi ở ví dụ 7
- Lập bảng viết rõ hơn sự so sánh phương pháp chia đôi và phương pháp tiếp tuyến ở ví dụ 7
- Sửa lại code sai trong ví dụ 19 tìm nghịch đảo của một số
- Nêu rõ cách kiểm tra điều kiện đầu vào của phương pháp tiếp tuyến và mối liên hệ với bài toán tìm min max
- Bổ sung kết quả của phương pháp *Newton* dùng công thức sai số mục tiêu ở ví dụ 7
- Bổ sung 7 ví dụ cơ bản minh họa cho các trường hợp dùng phương pháp *Newton*
- Bổ sung 4 bài toán thực tế đưa về giải phương trình bằng phương pháp *Newton*
- Sửa lại code phương pháp lai *Newt - Safe*
- Bổ sung ví dụ minh họa và kết quả chạy của phương pháp *Newt - Safe*
- Bổ sung phần tài liệu tham khảo

1 Lịch sử phương pháp

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Cái tên "phương pháp Newton" bắt nguồn từ mô tả của Issac Newton về một trường hợp đặc biệt của phương pháp trong *De analysis per aequationes numero terminorum infinitas* (viết năm 1669, xuất bản năm 1711) và trong *De metodis fluxionum et serierum infiniarum*. Tuy nhiên, phương pháp của ông khác về cơ bản so với phương pháp hiện đại được đưa ra ở trên: Newton chỉ áp dụng phương pháp này cho đa thức. Ông không tính các xấp xỉ x_n liên tiếp, mà tính một dãy các đa thức, và cuối cùng mới đưa ra một xấp xỉ cho nghiệm x. Cuối cùng, Newton xem phương pháp này hoàn toàn là đại số và không đề cập đến mối liên hệ với giải tích. Newton có thể đã sáng tạo ra một phương pháp tương tự nhưng kém chính xác hơn của Vieta. Phương pháp của Vieta có thể được tìm thấy trong công trình của nhà toán học Ba Tư Sharaf al-Din al-Tusi, khi mà học trò của ông đã sử dụng một dạng của phương pháp Newton để giải $x^p - N = 0$ tìm căn của N (1995). Một trường hợp đặc biệt của phương pháp Newton để tính căn bậc hai đã được biết đến từ thời cổ đại và thường được gọi là phương pháp Babylon.

Phương pháp của Newton được nhà toán học Nhật Bản thế kỷ 17 Seki Kowa sử dụng để giải các phương trình đơn biến, mặc dù thiếu sự liên kết với các kiến thức giải tích.

Phương pháp Newton được xuất bản lần đầu năm 1685 trong *A Treatise of Algebra both Historical and Practical* của John Wallis. Năm 1690, Joseph Raphson đã cho xuất bản một mô tả đơn giản trong *Analysis aequationum universalis*. Raphson một lần nữa chỉ xem phương pháp của Newton hoàn toàn là phương pháp đại số và hạn chế sử dụng nó cho đa thức, nhưng ông mô tả phương pháp theo các xấp xỉ liên tiếp thay vì dãy đa thức phức tạp của Newton. Cuối cùng, vào năm 1740, Thomas Simpson đã mô tả phương pháp của Newton như một phương pháp lặp để giải các phương trình phi tuyến tổng quát bằng giải tích, về cơ bản đưa ra các mô tả ở trên. Trong cùng một ấn phẩm, Simpson cũng đưa ra tổng quát cho hệ hai phương trình và phát biểu rằng phương pháp của Newton có thể được sử dụng để giải các bài toán tối ưu bằng cách cho gradient bằng 0.

Arthur Cayley, vào năm 1879, trong *The Newton - Fourier imaginary problem*, là người đầu tiên nhận thấy những khó khăn trong việc tổng quát phương pháp của Newton giải nghiệm phức của đa thức có bậc lớn hơn 2 và xấp xỉ đầu phức. Điều này đã mở đường cho việc nghiên cứu lý thuyết lặp của các hàm hữu tỷ.

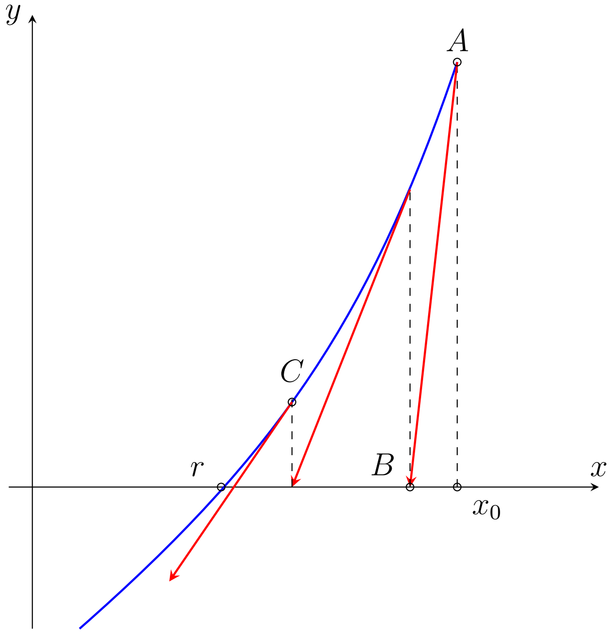
2 Xây dựng thuật toán

Phương pháp giải tích

Xét bài toán giải phương trình
$$f(x) = 0 \tag{1}$$
Gọi r là một nghiệm chính xác của 1, nghĩa là $f(r) = 0$.
Giả sử x_0 là một xấp xỉ của r. Xuất phát từ khai triển *Taylor* của $f(x)$ trong lân cận của x_0, ta có
$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2!}f''(\xi_0)$$
với ξ_0 nằm giữa x và x_0.
Giả sử f' và f'' tồn tại và liên tục trong lân cận điểm x_0. Nếu x_0 đủ gần nghiệm r của f và $f'(x_0)$ không quá lớn, thì hàm
$$g(x) = f(x_0) + f'(x_0)(x - x_0)$$
cho ta một xấp xỉ khá tốt của f trong lận cận của r. Trong trường hợp này, ta hoàn toàn có thể coi nghiệm của phương trình $g = 0$ là một xấp xỉ của r. Giải phương trình này ta có
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$
là một xấp xỉ tốt hơn cho r so với x_0.
Tương tự với x_1, sử dụng khai triển *Taylor* của $f(x)$ trong lân cận của r, ta có hàm
$$g_1(x) = f(x_1) + f'(x_1)(x - x_1)$$
là một xấp xỉ khá tốt của f trong lân cận của r. Giải phương trình $g_1(x) = 0$ ta lại có
$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$
là xấp xỉ tốt hơn cho r so với x_1.
Cứ như vậy, ta đã xây dựng được dãy $\{x_n\}_{n\geq 0}$
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n \geq 0 \tag{2}$$
Công thức 2 là công thức của phương pháp *Newton Raphson*, hay còn gọi là phương pháp tiếp tuyến.

1. Ta thấy nếu đạo hàm $f'(x)$ có độ lớn càng lớn trong lân cận của r thì lượng ta phải cộng thêm vào xấp xỉ x_n để có được x_{n+1} càng nhỏ. *Vì thế phương pháp tiếp tuyến đặc biệt tiện lợi khi đồ thị của hàm dốc trong lân cận của nghiệm r.*
2. Nếu ngược lại, độ lớn của đạo hàm $f'(x)$ nhỏ trong lân cận của nghiệm, thì độ lớn của $h_i = -\dfrac{f(x_i)}{f'(x_i)}$ sẽ lớn và việc tính toán bằng phương pháp này sẽ chậm và trong nhiều trường hợp, nó có thể không thành công. *Phương pháp này không nên được sử dụng trong những bài toán có hàm $f(x)$ gần như nằm ngang trong lân cận nghiệm.*
3. Quá trình lặp sẽ thất bại nếu $f'(x) = 0$ trong lân cận của r
4. Nếu f' không thay đổi nhiều, ta có thể xấp xỉ $f'(x_n)$ bởi $f'(x_0)$ và dùng công thức
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_0)}, \quad n \geq 0$$
Nhưng đổi lại, tốc độ hội tụ sẽ chậm hơn rất nhiều. Điều này sẽ được đề cập ở phần tính đạo hàm trong ví dụ 7

Phương pháp hình học



Hình 1: Xây dựng phương pháp Newton Raphson

Ý tưởng của phương pháp hình học là xuất phát từ điểm x_0 gần nghiệm chính xác r, ta sẽ xấp xỉ hàm f bằng đường tiếp tuyến với đồ thị của hàm tại x_0 để xác định một dãy lặp $\{x_n\}$ tiến dần về nghiệm r của phương trình $f(x) = 0$.
Tại điểm $A\left(x_0, f(x_0)\right)$ nằm trên đồ thị hàm f, vẽ đường tiếp tuyến với đồ thị có phương trình

$$y - f(x_0) = f'(x_0)(x - x_0)$$

Cho tiếp tuyến cắt trục hoành tại $B(x_1, 0)$, ta có

$$0 - f(x_0) = f'(x_0)(x_1 - x_0)$$

Suy ra $x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$. Chọn x_1 là xấp xỉ mới thì từ hình vẽ, rõ ràng x_1 tốt hơn x_0.
Từ $C\left(x_1, f(x_1)\right)$, kẻ tiếp tuyến với đồ thị hàm số, cắt trục hoành tại điểm $(x_2, 0)$ ta được x_2 là xấp xỉ tiếp theo tốt hơn x_1.
Tiếp tục quá trình này, ta có một dãy $\{x_n\}$ tiến dần về nghiệm r của phương trình $f(x) = 0$:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Đây chính là công thức 2.

1. Tiếp cận theo phương pháp hình học, ta giải thích được lý do vì sao phương pháp này được gọi là phương pháp tiếp tuyến.
2. Ta thấy rằng, một cách không hình thức, phương pháp *Newton* hội tụ với những hàm có hình dạng như trong hình vẽ.
3. Ngoài ra, trong hình vẽ trên, dãy $x_0, x_1, \ldots, x_n, \ldots$ bị chặn dưới bởi nghiệm đúng r.

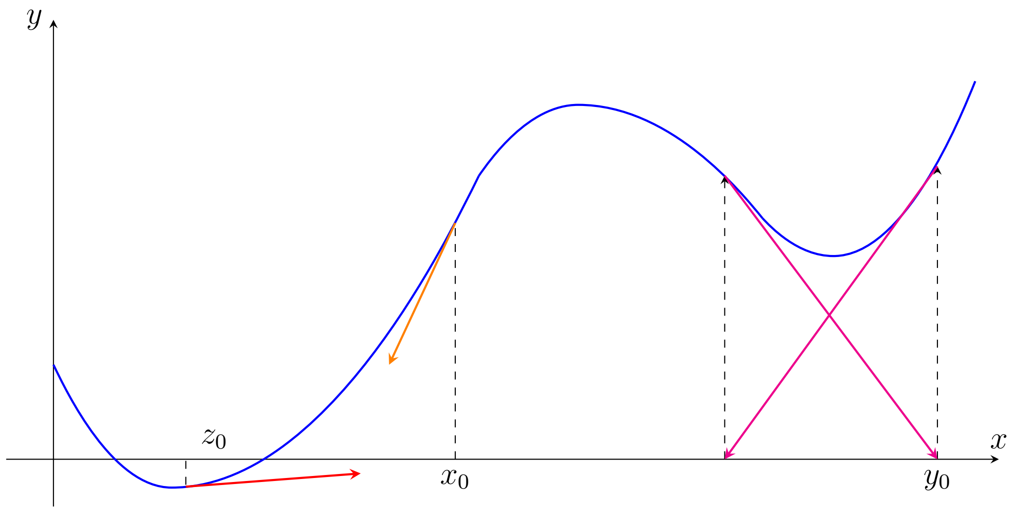
3 Chứng minh sự hội tụ

Xét phương trình

$$f(x) = 0$$

Phương pháp *Newton* với đầu vào là hàm $f(x)$ và xấp xỉ đầu x_0 của nghiệm r, sẽ sinh ra một dãy $x_0, x_1, \ldots, x_n, \ldots$ hội tụ về r.

1. Phương pháp này không phải luôn hội tụ với mọi cách chọn x_0.
2. Nếu xấp xỉ đầu chọn đủ gần với nghiệm chính xác r, thì dãy $\{x_n\}$ hội tụ rất nhanh, ta sẽ có được kết quả một cách nhanh chóng.
3. Nếu điểm x_0 không đủ gần r, hoặc x_0 chọn sai, thì dãy lặp có thể không hội tụ hoặc rơi vào vòng lặp vô hạn.
4. Có cách nào để biết điểm x_0 ta chọn là hợp lý ngoài việc dựa vào kết quả hội tụ, phân kì của dãy lặp?



Hình 2: Một số trường hợp chọn xấp xỉ đầu

Trong hình vẽ này

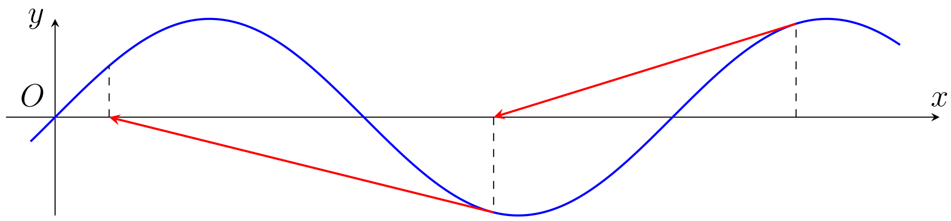
- Nếu chọn x_0 làm xấp xỉ đầu thì dãy lặp sẽ hội tụ rất nhanh.
- Nếu chọn y_0 làm xấp xỉ đầu thì sẽ bị rơi vào vòng lặp vô hạn.
- Nếu xuất phát từ z_0 thì dãy lặp sẽ phân kì vì $f'(z_0)$ rất nhỏ. Khi đó máy tính không thể tính được z_1 do giao điểm của tiếp tuyến với trục hoành ở xa vô cùng.

Xét một số trường hợp điển hình phương pháp sẽ không thành công. Code chương trình phương pháp *Newton* của các ví dụ này xem ở 11.

Ví dụ 1

Trường hợp hàm $f(x)$ dao động với biên độ không đổi và có nhiều nghiệm. Nếu ta chọn x_0 không đủ gần nghiệm ta muốn tìm thì dãy $\{x_n\}$ có thể hội tụ về nghiệm khác.

Giải phương trình $\sin x = 0$, nếu ta chọn $x_0 = 2.4\pi = 7.539822$ làm xấp xỉ đầu thì dãy lặp sẽ hội tụ về $x = 0$ như bảng dưới đây. Mặc dù từ hình vẽ, rõ ràng là ta muốn tìm nghiệm $x = 2\pi = 6.2831853$

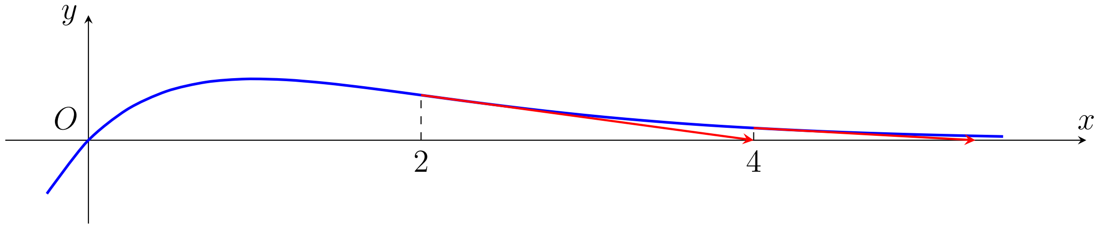


Hình 3: Trường hợp $y = \sin x$ và $x_0 = 2.4\pi$

Lần lặp	x_i	$f(x_i)$
1	4.4621423	-0.968851
2	0.5499853	0.522599
3	-0.0630409	-0.063008
4	8.376×10^{-4}	8.375×10^{-5}
5	-1.95×10^{-13}	-1.95×10^{-13}

Ví dụ 2

Trường hợp dãy lặp phân kì. Xét hàm $f(x) = xe^{-x}$ và $x_0 = 2$.



Hình 4: Trường hợp $y = xe^{-x}$ và $x_0 = 2$

Từ hình vẽ, ta nhận xét rằng các đường tiếp tuyến tại $\big(x_i, f(x_i)\big)$ cắt trục hoành tại các điểm ngày càng xa nghiệm đúng. Dãy không có dấu hiệu hội tụ.
Bảng dãy lặp của phương pháp:

Lần lặp	x_i	$f(x_i)$
1	4.000000	0.073262
2	5.333333	0.025749
3	6.564102	0.009255
4	7.743826	0.003356
...	...	...
15	19.72354	5.35989×10^{-8}

Ví dụ 3

Giải phương trình $f(x) = x^3 - 0.03x^2 + 2.4 \times 10^{-6} = 0$ bằng phương pháp tiếp tuyến

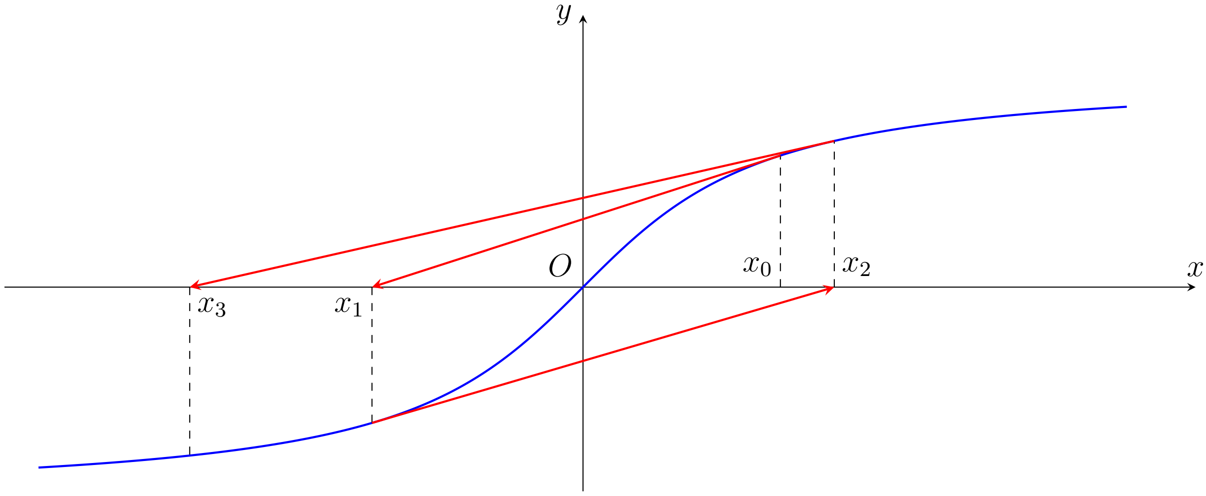
Thay vào công thức 2 ta thu được

$$x_{n+1} = x_n - \frac{x_n^3 - 0.03x_n^2 + 2.4 \times 10^{-6}}{3x_n^2 - 0.06x_n}$$

Dễ thấy $f'(x) = 3x^2 - 0.06x$ có nghiệm $x = 0$ và $x = 0.02$. Nếu chọn $x_0 = 0$ hoặc $x_0 = 0.02$ thì ta rơi vào trường hợp "chia cho 0", vì $f'(x_0) = 0$. Trong trường hợp này, phương pháp không thực hiện được.

Ví dụ 4

Dãy lặp tạo thành dạng "cycle" không bao giờ hội tụ. Xét hàm $f(x) = \tan^{-1}(x)$ và $x_0 = 1.45$.

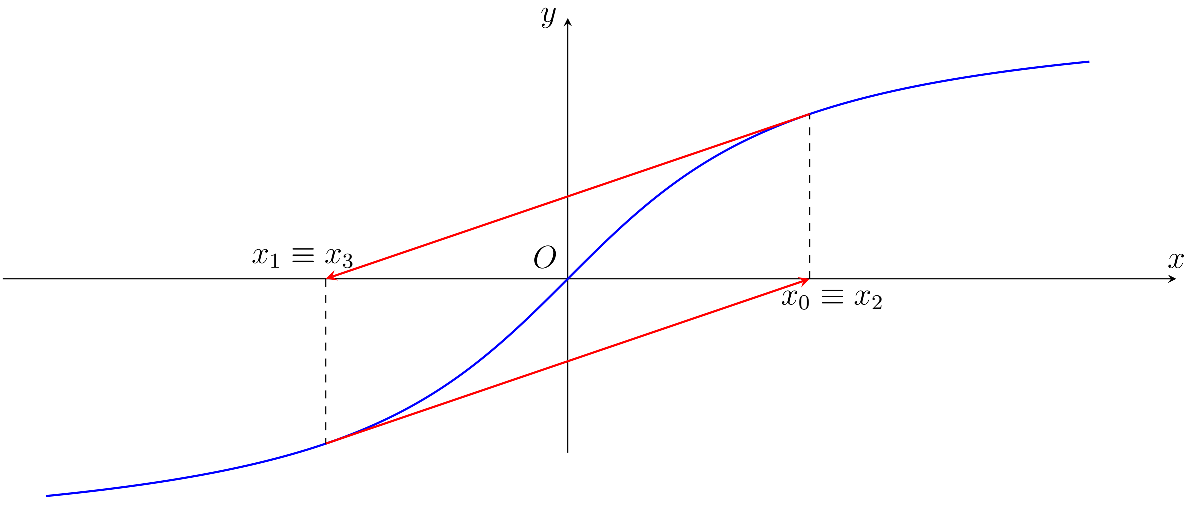


Hình 5: Trường hợp $y = \tan^{-1} x$ và $x_0 = 4.5$

Từ bảng và hình vẽ, ta thấy rằng trong trường hợp này dãy không hội tụ đến nghiệm $x = 0$

Lần lặp	x_i	$f(x_i)$
1	-1.550263	-0.997907
2	1.845931	1.074323
3	-2.889107	-1.237575
4	8.674496	1.456074
5	-102.4425	-1.561035
6	16281.33	1.570734
...	...	...

Tương tự, nếu chọn $x_0 = 1.391745008$, ta cũng rơi vào vòng lặp vô hạn.



Hình 6: Trường hợp $y = \tan^{-1} x$ và $x_0 = 1.391745008$

Qua các ví dụ trên, ta thấy rằng không phải lúc nào phương pháp tiếp tuyến cũng hội tụ. Dãy lặp chỉ hội tụ về nghiệm đúng nếu ta chọn được x_0 "hợp lý". Nhưng có phải lúc nào cũng tồn tại một xuất phát điểm x_0 để đảm bảo cho sự hội tụ của phương pháp? Định lý sau sẽ cho ta câu trả lời

Định lý 1

Nếu f, f' và f'' liên tục trong lân cận của điểm "không" r của f và nếu $f'(r) \neq 0$, thì tồn tại $\delta > 0$ với tính chất sau: Nếu xấp xỉ đầu x_0 của phương pháp $Newton$ thỏa mãn $|x_0 - r| \leq \delta$, thì mọi điểm x_n đều thỏa mãn bất đẳng thức này, và chúng hội tụ đến r, với tốc độ hội tụ bậc 2, nghĩa là

$$|r - x_{n+1}| \leq c(\delta)|r - x_n|^2$$

ở đó $c(\delta)$ cho bởi biểu thức 4 dưới đây.

Chứng minh. Đặt $e_n = r - x_n$. Từ công thức xác định x_n theo phương pháp $Newton$ ta có

$$e_{n+1} = r - x_{n+1} = r - x_n + \frac{f(x_n)}{f'(x_n)} = e_n + \frac{f(x_n)}{f'(x_n)} = \frac{e_n f'(x_n) + f(x_n)}{f'(x_n)}$$

Theo công thức khai triển $Taylor$, tồn tại một điểm ξ_n nằm giữa x_n và r sao cho

$$0 = f(r) = f(x_n + e_n) = f(x_n) + e_n f'(x_n) + \frac{1}{2}e_n^2 f''(\xi_n)$$

Ta viết lại thành

$$e_n f'(x_n) + f(x_n) = -\frac{1}{2}e_n^2 f''(\xi_n)$$

Thay lên phương trình đầu tiên ta có

$$e_{n+1} = -\frac{1}{2}\left(\frac{f''(\xi_n)}{f'(x_n)}\right)e_n^2 \tag{3}$$

Từ đây, ta định nghĩa hàm

$$c(\delta) = \frac{1}{2}\frac{\max\limits_{|x-r|\leq\delta}\left|f''(x)\right|}{\min\limits_{|x-r|\leq\delta}\left|f'(x)\right|}, \quad (\delta > 0) \tag{4}$$

Dễ thấy rằng với bất kì 2 điểm x và ξ cách nghiệm r một khoảng không quá δ, thì ta sẽ có $\frac{1}{2}\left|\frac{f''(\xi)}{f'(x)}\right| \leq c(\delta)$. Chọn δ thỏa mãn $\delta c(\delta) < 1$. Điều này hoàn toàn có thể làm được, vì khi δ tiến đến 0, thì $\lim c(\delta) = \frac{1}{2}\left|\frac{f''(r)}{f'(r)}\right|$, và do đó $\lim \delta c(\delta) = 0$, chú ý rằng $f'(r) \neq 0$.

Đặt $\rho = \delta c(\delta)$. Cố định $\delta, c(\delta), \rho$ với $\rho < 1$. Giả sử ở lần lặp nào đó x_n nằm cách r không quá δ, ta có

$$|e_n| = |r - x_n| \leq \delta \quad \text{và} \quad |\xi_n - r| \leq \delta$$

Theo định nghĩa $c(\delta)$, ta có $\frac{1}{2}\left|\frac{f''(\xi_n)}{f'(x_n)}\right| \leq c(\delta)$, từ 3 thu được

$$|e_{n+1}| = \frac{1}{2}\left|\frac{f''(\xi_n)}{f'(x_n)}\right| e_n^2 \leq c(\delta)e_n^2 \leq \delta c(\delta)|e_n| = \rho|e_n|$$

Hệ quả là x_{n+1} cũng nằm trong hình tròn tâm r bán kính δ, bởi vì

$$|r - x_{n+1}| = |e_{n+1}| \leq \rho|e_n| \leq |e_n| \leq \delta$$

Do đó, nếu chọn xấp xỉ đầu x_0 cách r không quá δ, thì

$$|e_n| \leq \rho|e_{n-1}| \leq \rho^2|e_{n-2}| \leq \ldots \leq \rho^n|e_0|$$

Vì $0 < \rho < 1$ nên $\lim\limits_{n\to\infty} \rho^n = 0$ và $\lim\limits_{n\to\infty} e_n = 0$. Nói cách khác, ta thu được

$$\lim_{n\to\infty} x_n = r$$

Trong quá trình lặp này, ta có $|e_{n+1}| \le c(\delta)|e_n|$ □

1. r là nghiệm đơn, có nghĩa là $f(r) = 0$ và $f'(r) \neq 0$
2. Trên đoạn $[r-\delta, r+\delta]$ ta định nghĩa $c(\delta)$ theo 4. Điều này có nghĩa là

$$\min_{|x-r|\le\delta} \left|f'(x)\right| > 0$$

hay f' khác 0 và không đổi dấu trong lân cận δ của r. Do đó khoảng $(r-\delta, r+\delta)$ là khoảng phân ly nghiệm r của hàm f.
3. Như vậy, nếu r là nghiệm đơn của f thì sẽ luôn tồn tại một lân cận δ quanh r mà không chứa nghiệm nào khác của f sao cho với mọi $x_0 \in [r-\delta, r+\delta]$, phương pháp $Newton$ hội tụ đến r với tốc độ hội tụ bậc hai.
4. Nhưng δ chọn như thế nào là đủ nhỏ? Định lý trên chưa cho ta ngay câu trả lời, nhưng nó gợi ý cho ta kết quả sau

Định lý 2

Giả sử hàm f có f', f'' liên tục và có nghiệm đơn r trong khoảng (a, b) nào đó. Định nghĩa tỉ số

$$M = \frac{1}{2} \frac{\max\limits_{x\in[a,b]} \left|f''(x)\right|}{\min\limits_{x\in[a,b]} \left|f'(x)\right|}$$

và giả sử $M < \infty$. Khi đó, với mọi x_0 thỏa mãn $M|r - x_0| < 1$, dãy lặp của phương pháp $Newton$: dãy $\{x_n\}$ hội tụ.

Chứng minh. Xuất phát từ ý tưởng định lý trên, xét

$$|e_{n+1}| = -\frac{1}{2}\left(\frac{f''(\xi_n)}{f'(x_n)}\right) e_n^2 \le M e_n^2, \quad n \ge 0$$

Với $n = 0$ ta có $|e_1| \le M e_0^2$
Với $n = 1$ ta có

$$|e_2| \le M e_1^2 \le M\left(M e_0^2\right)^2 = \frac{1}{M}\left(M e_0\right)^4$$

Với $n = 3$ ta có

$$|e_3| \le M e_2^2 \le M\left[\frac{1}{M}\left(M e_0\right)^4\right]^2 = \frac{1}{M}\left(M e_0\right)^8$$

Như vậy, bằng quy nạp ta có

$$|e_n| \le \frac{1}{M}\left(M e_0\right)^{2^n}$$

Vì $M|r - x_0| < 1$ nên $\lim\limits_{n\to\infty} e_n = 0$, suy ra dãy $\{x_n\}$ hội tụ. □

1. Ta đang xét (a, b) là khoảng phân ly nghiệm r của f. Chỉ khi đó, ta mới định nghĩa được M và có $M < \infty$. Chú ý $M > 0$.
2. Nếu r là nghiệm đơn trong khoảng phân ly (a, b), ta có thể chọn $\delta < \dfrac{1}{M}$.
3. Nếu M nhỏ, khoảng chọn x_0 sẽ rộng, tối đa là khoảng (a, b). Nếu M lớn, khoảng chọn x_0 sẽ hẹp hơn.
4. Định lý trên chỉ là điều kiện đủ. Nếu chọn x_0 có $M|r - x_0| \geq 1$, ta không thể suy ra dãy $\{x_n\}$ hội tụ, nhưng cũng không có căn cứ để kết luận nó phân kì.

Ta xét ví dụ áp dụng thuật toán *Newton*.

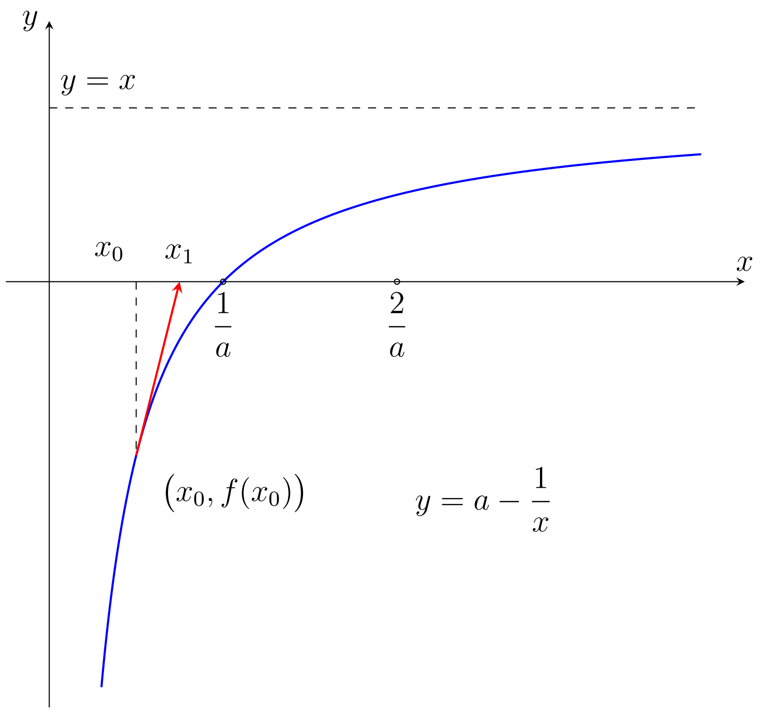
Ví dụ 5

Tìm nghịch đảo của một số. Xét phương trình

$$f(x) = a - \frac{1}{x} = 0 \quad \text{với } a > 0 \text{ cho trước}$$

Sử dụng công thức 2, ta có

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n\left(2 - ax_n\right) \quad n \geq 0$$



Hình 7: Tìm nghịch đảo của một số bằng phương pháp Newton

1. Ta nhận xét rằng công thức của dãy lặp không chứa phép chia.
2. Điều đó làm cho bài toán này có tính ứng dụng giúp máy tính thực hiện phép chia mà không cần định nghĩa phép toán chia!
3. Ứng dụng này sớm được sử dụng trong những thế hệ máy tính cũ, và trong các máy tính hiện đại, thuật toán này cũng góp mặt và là một phần quan trọng trong thuật toán chia.

Ta phân tích sự hội tụ của thuật toán, tốc độ chạy của nó, cũng như sự phụ thuộc vào giá trị đầu x_0. Đầu tiên

$$\frac{1}{a} - x_n = \frac{1}{a} - 2x_{n-1} + ax_{n-1}^2 = a\left(\frac{1}{a} - x_{n-1}\right)^2$$

Suy ra

$$\frac{1}{a} - x_n = a^{2^n - 1}\left(\frac{1}{a} - x_0\right)^{2^n} = \frac{1}{a}(1 - ax_0)^{2^n}$$

Từ công thức này, ta thấy rằng để sai lệch $e_n = \frac{1}{a} - x_n$ hội tụ đến 0 khi $n \to \infty$, điều kiện cần và đủ là

$$-1 < 1 - ax_0 < 1$$

tương đương với

$$0 < x_0 < \frac{2}{a}$$

Như vậy, nếu x_0 chọn thỏa mãn điều kiện này, thì dãy $\{x_n\}$ hội tụ đến $\frac{1}{a}$
Từ công thức $e_{n+1} = ae_n^2$, ta thấy rằng thuật toán hội tụ với tốc độ là bậc hai.

1. Ta thấy rằng thuật toán này sẽ hội tụ với mọi xấp xỉ đầu x_0 thỏa mãn $0 < x_0 < \frac{2}{a}$. Tuy nhiên, trong trường hợp $a < 1$, ta có thể chọn chính a làm xấp xỉ đầu. Xét ví dụ $a = 10^{-10}$. Khi đó,

$$x_1 = 2.10^{-10} + 10^{-30} \simeq 2.10^{-10}$$

Lần lặp thứ nhất cho ta một số cỡ gấp đôi xấp xỉ đầu. Tương tự, x_2 xấp xỉ hai lần x_1. Quá trình này cứ tiếp tục cho đến khi $x_k \simeq 10^{10}$. Do đó, ta phải có $2^k.10^{-10} \simeq 10^{10}$, hay $k \simeq 66$! Điều đó có phải là quá nhiều chỉ để tính nghịch đảo của một số?
2. Điều đó không có nghĩa là dãy lặp không tốt, chỉ là nó cần một xấp xỉ đầu tốt hơn! Giả sử a được viết dưới dạng số nhị phân $a = 2^m x_1$, ở đó m nguyên và $\frac{1}{2} \leq x_1 < 1$. Khi đó, nếu chọn $x_0 = 2^{-m}$ thì ta có $ax_0 = 2^m x_1 .2^{-m} = x_1$ và

$$\frac{1}{2} \leq ax_0 < 1$$

Bất đẳng thức này chặt hơn bất đẳng thức ở trên. Đồng thời, thay vào công thức bên trên, ta thấy được thuật toán hội tụ rất nhanh

$$\left|\frac{1}{a} - x_n\right| \leq \frac{1}{a}\left(\frac{1}{2}\right)^{2^n} \leq 2x_0\left(\frac{1}{2}\right)^{2^n}$$

Ta chú ý đến kết quả sau.

Định lý 3

Giả sử (a, b) là khoảng phân ly nghiệm r của phương trình 1. $f(x)$ liên tục, có đạo hàm liên tục đến cấp 2, hơn nữa $f'(x), f''(x)$ khác 0 và không đổi dấu trên miền $a \leq x \leq b$, thì nếu xấp xỉ đầu x_0 được chọn thỏa mãn

$$f(x_0)f''(x_0) > 0 \tag{5}$$

dãy $\{x_n\}$ tính theo công thức 2 sẽ hội tụ đơn điệu về r.

Chứng minh. Không mất tính tổng quát, giả sử $f(a) < 0, f(b) > 0, f'(x) > 0, f''(x) > 0$ với $a \leq x \leq b$ (các trường hợp còn lại chứng minh tương tự). Từ giả thiết ta có $f(x_0) > 0$ (lấy luôn $x_0 = b$)
Bằng quy nạp, ta chứng minh được $x_n > r$ $(n = 0, 1, 2, \ldots)$, và do đó $f(x_n) > 0$. Thật vậy,. trước hết, có $x_0 > r$. Giả sử $x_n > r$. Đặt

$$r = x_n + (r - x_n)$$

Sử dụng khai triển $Taylor$, ta có

$$0 = f(r) = f(x_n) + f'(x_n)(r - x_n) + \frac{1}{2} f''(\xi_n)(r - x_n)^2$$

ở đó $r < \xi_n < x_n$.
Vì $f''(x) > 0$ nên

$$f(x_n) + f'(x_n)(r - x_n) < 0$$

suy ra

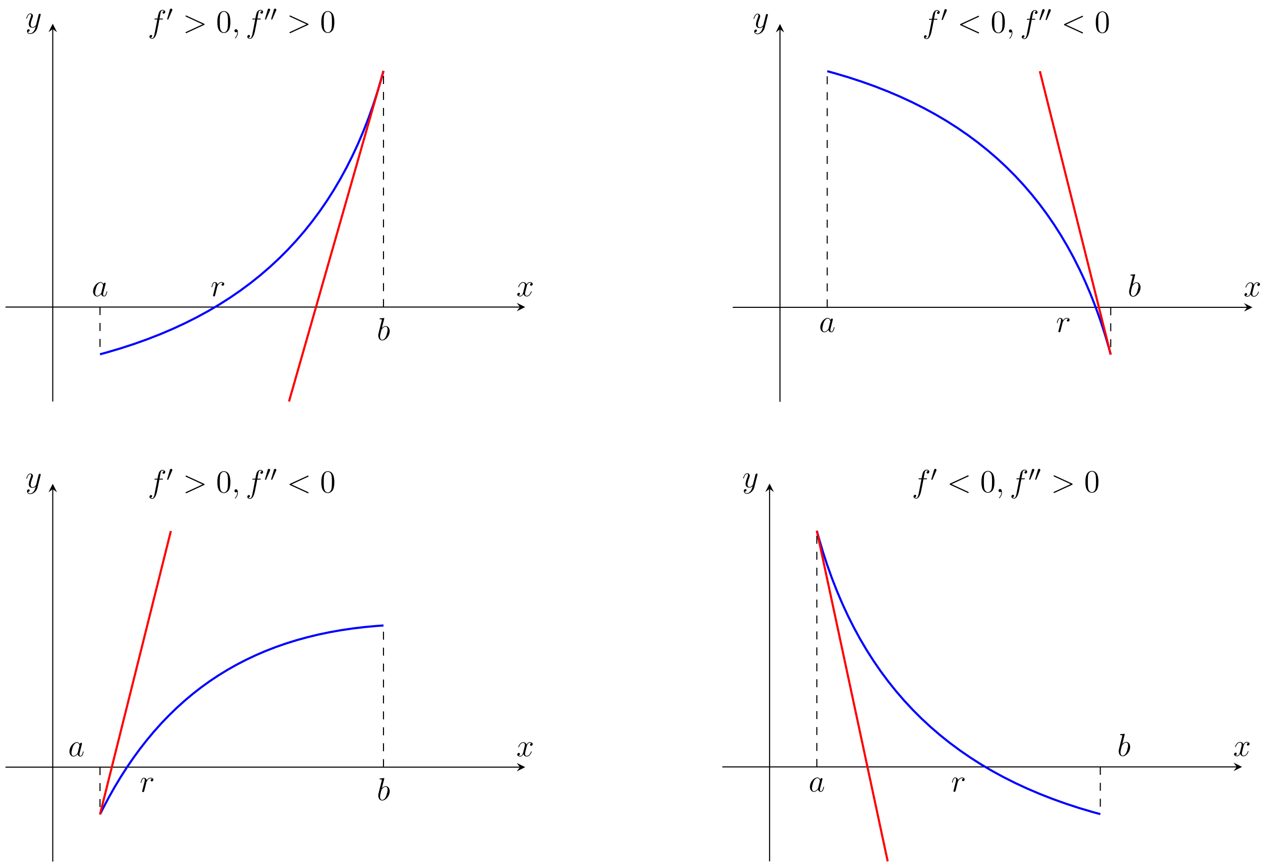
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} > r$$

Từ dấu của $f(x_n), f'(x_n)$ ta có $x_{n+1} < x_n$ với $n \geq 0$. Như vậy, $\{x_n\}$ là dãy giảm và bị chặn dưới, nên nó có giới hạn. Gọi $\overline{r} = \lim\limits_{n \to \infty} x_n$, thay vào 2 ta có

$$\overline{r} = \overline{r} - \frac{f(\overline{r})}{f'(\overline{r})}$$

suy ra $f(\overline{r}) = 0$, hay $\overline{r} = r$. □

1. Ta sử dụng định lý này để lập trình chương trình phương pháp $Newton$ vì khoảng (a, b) là tương đối dễ xác định, đặc biệt là với các các công cụ hiện đại như Geogebra, Matlab, Maple, Wolfram Alpha,... Đồng thời ta cũng không cần phải đi tìm M như định lý 3.
2. r là nghiệm duy nhất trên (a, b). Trong phép chứng minh định lý, để $\dfrac{f(\overline{r})}{f'(\overline{r})}$ tồn tại, ta phải có $f'(\overline{r}) \neq 0$, hay r là nghiệm đơn của f.
3. Ta chú ý chọn điểm x_0 thỏa mãn 5, nếu không giao điểm của tiếp tuyến với trục hoành sẽ nằm ngoài đoạn $[a, b]$, quá trình lặp có thể sẽ không hội tụ.
4. Đặc biệt, kiểm tra điều kiện đầu vào của định lý, f', f'' không đổi dấu trên $[a, b]$ không phải là một bài toán lập trình dễ, nó đòi hỏi công việc thậm chí còn nhiều hơn chính bản thân phương pháp $Newton$. Ta sẽ phải xây dựng những hàm riêng để kiểm tra điều kiện này.
5. Điều kiện áp dụng đặt lên phương pháp theo định lý này rất chặt, giống với điều kiện sử dụng phương pháp dây cung. Có thể hình dung ở đây nếu $f' = 0$, tiếp tuyến với đồ thị hàm số sẽ không cắt trục hoành, nếu $f'' = 0$, ta có thể rơi vào vòng lặp vô hạn.



Hình 8: Bốn trường hợp cơ bản áp dụng định lý 3

Tiếp cận trên phương diện sử dụng phương pháp lặp đơn

Nhắc lại **điều kiện đủ** về sự hội tụ của phương pháp lặp đơn.

Định lý 4

Cho g là một hàm liên tục trên đoạn đóng $[a, b]$ với $g\colon [a, b] \longrightarrow [a, b]$. Hơn nữa, giả sử g có đạo hàm trên khoảng (a, b) và tồn tại hằng số dương $k < 1$ sao cho

$$\left|g'(x)\right| \leq k < 1$$

với mọi $x \in (a, b)$. Khi đó, dãy $\{x_n\}$ cho bởi $x_{n+1} = g(x_n)$ hội tụ đến điểm bất động r (duy nhất trên đoạn này) với mọi $x_0 \in [a, b]$.

Ta sẽ chứng minh lại sự hội tụ của phương pháp *Newton* đã được chứng mình ở định lý 3.

Định lý 5

Giả sử f có đạo hàm liên tục đến cấp 2 trong đoạn $[a, b]$ với $r \in (a, b)$ thỏa mãn $f(r) = 0$. Hơn nữa, giả sử $f'(r) \neq 0$. Khi đó, tồn tại $\delta > 0$ sao cho với mọi $x_0 \in I = [r - \delta, r + \delta]$, dãy $\{x_n\}$ cho bởi công thức 2 hội tụ đến r.

Chứng minh. Hàm lặp *Newton* cho bởi

$$g(x) = f - \frac{f(x)}{f'(x)}$$

Điều kiện $f(r) = 0$ cho ta $g(r) = r$, hay r là điểm bất động của hàm g. Ta chứng minh định lý trên theo 3 bước

Bước 1. Chứng minh g liên tục "xung quanh" r.
Với định nghĩa hàm lặp g và tính liên tục của f, điểm gián đoạn có thể có của g chỉ có thể là do $f' = 0$. Nhưng để ý rằng, vì f' liên tục và giả thiết $f'(r) \neq 0$, suy ra tồn tại δ_1 sao cho $f'(x) \neq 0$ với mọi $x \in I_1 = [r - \delta_1, r + \delta_1] \subset [a, b]$. Kết luận này có được bởi cho dù f' có thể có nhiều nghiệm đâu đó trong lân cận của r, tính liên tục của nó đòi hỏi khoảng cách giữa r và nghiệm nào đó chỉ là hữu hạn. Do vậy, g xác định, và liên tục trên I_1

Bước 2. Chứng minh $\left|g'(x)\right|$ nhỏ "xung quanh" r.
Tính toán trực tiếp ta có

$$g'(x) = \frac{f(x)f''(x)}{\left[f'(x)\right]^2}$$

Ta đã chứng minh ở trên $f'(x) \neq 0$ với mọi $x \in I_1$, suy ra $g'(x)$ liên tục trên I_1. Hơn thế nữa, ta có $g'(r) = 0$. Chọn số k bất kì thỏa mãn $0 < k < 1$. Tương tự cách làm ở bước 1, chỉ ra được tồn tại số dương δ, mà $\delta < \delta_1$, sao cho

$$\left|g'(x)\right| \leq k < 1$$

với mọi $x \in I = [r - \delta, r + \delta]$.

Bước 3. Chứng minh g đi từ đoạn I vào chính nó.
Lấy $x \in I$. Khi đó ta có

$$\begin{aligned}\left|g(x) - r\right| &= \left|g(x) - g(r)\right| \\ &= \left|g'(\xi)\right||x - r| && \text{với } \xi \text{ nằm giữa } x \text{ và } r \\ &\leq k|x - r| < |x - p| \leq \delta\end{aligned}$$

vì $x \in [r - \delta, r + \delta]$. Do đó, $g(x) \in [r - \delta, r + \delta] = I$. □

4 Công thức sai số

Định lý 6

Gọi r là nghiệm đúng và $\overline{x}$ là nghiệm xấp xỉ của phương trình $f(x) = 0$, cả hai cùng nằm trong đoạn $[a, b]$, giả sử $\left|f'(x)\right| \geq m_1 > 0$ với $a \leq x \leq b$. Khi đó ta có đánh giá sau

$$|\overline{x} - r| \leq \frac{\left|f(\overline{x})\right|}{m_1} \tag{6}$$

Chứng minh. Áp dụng định lý giá trị trung gian, ta có

$$f(\overline{x}) - f(r) = (\overline{x} - r)f'(c)$$

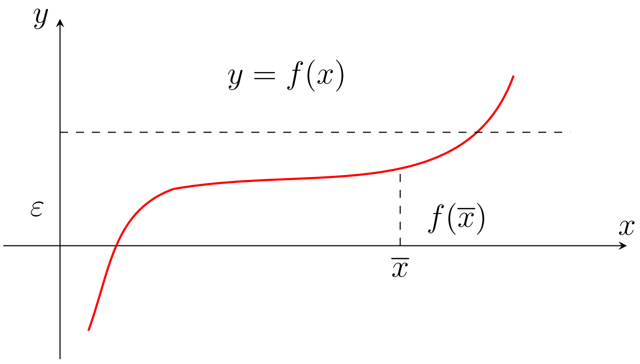
ở đó c nằm giữa $\overline{x}$ và r, cũng có nghĩa là $c \in (a, b)$. Từ đây, vì $f(r) = 0$ và $\left|f'(c)\right| \geq m_1$, ta thu được

$$\left|f(\overline{x}) - f(r)\right| = \left|f(\overline{x})\right| \geq m_1|\overline{x} - r|$$

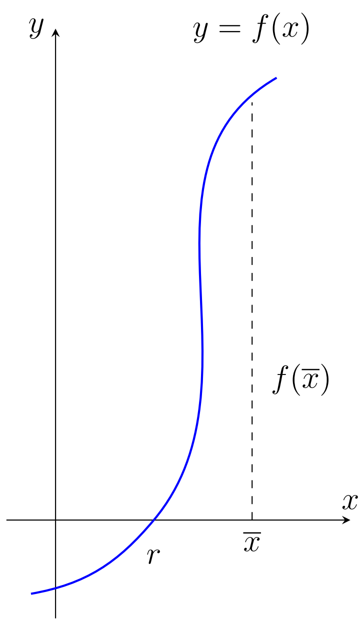
Do đó

$$|\overline{x} - r| \leq \frac{\left|f(\overline{x})\right|}{m_1}$$

□



Hình 9: Sai số nếu $f(\overline{x})$ nhỏ trong lân cận r



Hình 10: Sai số nếu $f(\overline{x})$ lớn trong lân cận r

1. Trong nhiều trường hợp, độ chính xác của $\overline{x}$ được ước lượng bằng mức độ thỏa mãn phương trình $f(x) = 0$, có nghĩa là nếu $\left|f(\overline{x})\right|$ càng nhỏ thì $\overline{x}$ càng chính xác.
2. Hai hình trên đây cho thấy cách tiếp cận này là sai. Nếu ta nhân $f(x) = 0$ với một số N tùy ý khác 0 thì ta được một phương trình tương đương. Khi đó $\left|Nf(\overline{x})\right|$ có thể làm to nhỏ tùy ý bằng cách chọn N
3. Trong công thức 6, nếu m_1 rất nhỏ thì khi $\left|f(\overline{x})\right|$ nhỏ, ta vẫn có thể có $|\overline{x} - r|$ lớn, trong trường hợp này $\overline{x}$ có thể chưa phải nghiệm đủ chính xác

Định lý 7

Gọi r là nghiệm đúng và x_n là nghiệm xấp xỉ của phương trình $f(x) = 0$ xác định bởi công thức 2, giả sử $\left|f'(x)\right| \geq m_1 > 0, \left|f''(x)\right| \leq M_2$ với $a \leq x \leq b$. Khi đó ta có đánh giá sau

$$|x_n - r| \leq \frac{M_2}{2m_1}|x_n - x_{n-1}|^2 \tag{7}$$

Chứng minh. Áp dụng khai triển *Taylor* ta có

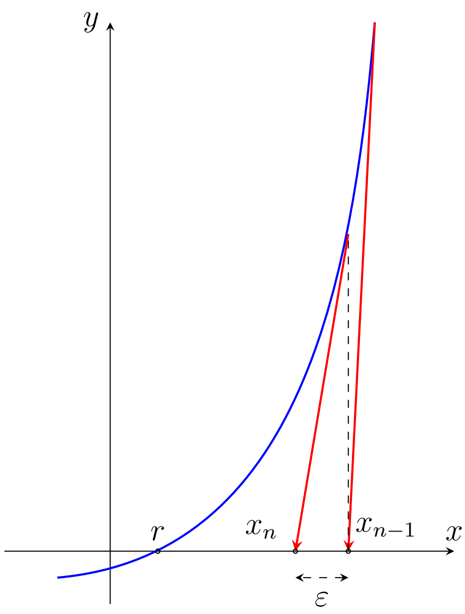
$$\begin{aligned} f(x_n) &= f\left[x_{n-1} + (x_n - x_{n-1})\right] \\ &= f(x_{n-1}) + f'(x_{n-1})(x_n - x_{n-1}) + \frac{1}{2}f''(\xi_{n-1})(x_n - x_{n-1})^2, \quad \xi_{n-1} \in (x_{n-1}, x_n) \end{aligned}$$

Từ 2 có $f(x_{n-1}) + f'(x_{n-1})(x_n - x_{n-1}) = 0$. Suy ra $\left|f(x_n)\right| \leq \dfrac{M_2}{2}(x_n - x_{n-1})^2$. Thay vào 6 ta thu được

$$|x_n - r| \leq \frac{M_2}{2m_1}|x_n - x_{n-1}|^2$$

□

1. Trong nhiều trường hợp, độ chính xác của x_n được ước lượng bằng khoảng cách giữa 2 xấp xỉ liên tiếp, có nghĩa là nếu $|x_n - x_{n-1}|$ càng nhỏ, hay x_n và x_{n-1} có càng nhiều chữ số trùng nhau, thì x_n càng chính xác.
2. Hình dưới đây cho thấy cách tiếp cận này là sai. Ngay cả khi hai xấp xỉ liên tiếp rất gần nhau, cũng không thể đảm bảo rằng ta đã thu được nghiệm với độ chính xác mong muốn, $|x_n - x_{n-1}| < \epsilon$ nhưng x_n chưa phải là một xấp xỉ tốt, nó ở rất xa nghiệm đúng r.
3. Trong công thức 7, nếu đại lượng $\dfrac{M_2}{2m_1}$ rất lớn, thì khi $|x_n - x_{n-1}|^2$ nhỏ, ta vẫn có thể có $|x_n - r|$ lớn. Đồng thời ta cũng chú ý rằng ở đây $|x_n - x_{n-1}|$ có mũ 2.



Hình 11: Sai số trong trường hợp ε nhỏ nhưng sai số tuyệt đối khá lớn

5 Tốc độ hội tụ

Trong bài toán giải số, ví dụ như bài toán giải một phương trình siêu việt, phương trình phi tuyến phức tạp, hay tính giá trị của một tích phân xác định,... phương pháp là người ta sẽ lập trình chương trình tạo ra một dãy các số thực $x_1, x_2, x_3, \ldots$ tiến dần đến đáp án chính xác (đáp án giải tích) của bài toán. Do đó, để mô tả sự hội tụ nhanh, chậm của một dãy, người ta sử dụng một thuật ngữ đặc biệt, tốc độ hội tụ.

Định nghĩa 1

Cho $\{x_n\}_{n\geq 0}$ là dãy số thực hội tụ có giới hạn $\lim\limits_{n\to\infty} x_n = r$. Ta nói dãy $\{x_n\}$ hội tụ với bậc $\alpha \geq 1$ đến điểm r nếu tồn tại số dương c và số $N \in \mathbb{N}$ thỏa mãn

$$|r - x_{n+1}| \leq c|r - x_n|^\alpha \quad n \geq N$$

Ta tạm gọi c là hằng số tiệm cận.

Thông thường, trong thực tế, ta hay gặp các tốc độ hội tụ cơ bản sau.
Ta nói dãy hội tụ **tuyến tính** nếu $\alpha = 1$ và $c < 1$.
Ta nói dãy hội tụ **superlinear** nếu tồn tại dãy $\{\varepsilon_n\}$ có $\lim\limits_{n\to\infty} \varepsilon_n = 0$ sao cho

$$|r - x_{n+1}| \leq \varepsilon|r - x_n| \quad n \geq N$$

Ta nói dãy hội tụ **bậc hai** nếu $\alpha = 2$. Trong trường hợp này, c không nhất thiết phải lớn hơn 1.
Ta nói kĩ hơn về tốc độ hội tụ bậc hai của phương pháp $Newton$, đã được chứng minh ở định lý 3.

Định lý 8

Gọi r là nghiệm đúng và x_n là nghiệm xấp xỉ của phương trình $f(x) = 0$ xác định bởi công thức 2, giả sử $\left|f'(x)\right| \geq m_1 > 0, \left|f''(x)\right| \leq M_2$ với $a \leq x \leq b$. Khi đó ta có đánh giá sau

$$|r - x_{n+1}| \leq \frac{M_2}{2m_1}|r - x_n|^2 \tag{8}$$

Chứng minh. Áp dụng khai triển $Taylor$ ta có

$$\begin{aligned} f(x_n) &= f\left[x_{n-1} + (x_n - x_{n-1})\right] \\ &= f(x_{n-1}) + f'(x_{n-1})(x_n - x_{n-1}) + \frac{1}{2}f''(\xi_{n-1})(x_n - x_{n-1})^2, \quad \xi_{n-1} \in (x_{n-1}, x_n) \end{aligned}$$

Suy ra

$$r = x_n - \frac{f(x_n)}{f'(x_n)} - \frac{1}{2}\frac{f''(\xi_n)}{f'(x_n)}(r - x_n)^2$$

Kết hợp với 2, ta thu được $r - x_{n+1} = -\frac{1}{2}\frac{f''(\xi_n)}{f'(x_n)}(r - x_n)^2$. Từ đó ta có

$$|r - x_{n+1}| \leq \frac{M_2}{2m_1}|r - x_n|^2$$

□

1. Đặc biệt, nếu $\dfrac{M_2}{2m_1} \leq 1$ và $|r - x_n| < 10^{-m}$, thì từ 8 ta có

$$|r - x_{n+1}| < 10^{-2m}$$

Trong trường hợp này, nếu x_n chính xác đến m chữ số thập phân, thì xấp xỉ tiếp theo x_{n+1} sẽ chính xác đến khoảng $2m$ chữ số thập phân. Có nghĩa là sau mỗi lần lặp của phương pháp *Newton*, số chữ số chính xác tăng lên gần như gấp đôi.

2. Giả sử $\dfrac{M_2}{2m_1} = 1$ và $e_0 = 10^{-1}$. Khi đó

$$e_1 \simeq 10^{-2}, e_2 \simeq 10^{-4}, e_3 \simeq 10^{-8}, e_4 \simeq 10^{-16}, \ldots$$

Ví dụ 6

Tính $\sqrt{10}$ với xấp xỉ đầu $x_0 = 3.0$.

Bảng sau cho ta các chữ số chính xác ở mỗi lần lặp

Lần lặp	x_i
0	3.
1	3.1666666666666667
2	3.1622807017543860
3	3.162277660169842
4	3.162277660168380

Ta thấy sau mỗi lần lặp số chữ số chính xác tăng lên gần như 2 lần.

Tiếp cận trên phương diện sử dụng phương pháp lặp đơn

Tiếp cận dựa trên tốc độ hội tụ của phương pháp lặp đơn giúp ta một cái nhìn rõ hơn về ưu điểm vượt trội của phương pháp *Newton*, hơn thế nữa, nó giúp ta mở rộng phương pháp này trong trường hợp nghiệm bội (điều này sẽ được bàn đến ở dưới sau)

- Ta đã biết rằng tốc độ hội tụ của phương pháp lặp đơn phụ thuộc vào các tính chất của hàm lặp $g(x)$.
- Phương pháp lặp đơn học trong chương trình có tốc độ hội tụ tuyến tính.
- Trong mục này, ta chỉ ra phương pháp *Newton* là trường hợp ta chọn hàm g một cách "khéo léo" để dãy lặp hội tụ với tốc độ nhanh hơn đáng kể

Định lý 9

Cho $g(x)$ là hàm liên tục trên đoạn đóng $[a, b]$ với $g\colon [a, b] \longrightarrow [a, b]$. Hơn thế nữa, giả sử g có đạo hàm trên khoảng (a, b) và tồn tại số dương $k < 1$ sao cho

$$\left|g'(x)\right| \leq k < 1$$

với mọi $x \in (a, b)$. Nếu $g'(r) \neq 0$, thì với mọi $x_0 \in [a, b]$, dãy $x_n = g(x_{n-1})$ hội tụ tuyến tính đến điểm bất động r

Định lý 10

Cho $g(x)$ là hàm liên tục trên đoạn đóng $[a, b]$ với đạo hàm đến cấp $\alpha > 1$ liên tục trên khoảng (a, b). Hơn nữa, gọi $r \in [a, b]$ là điểm bất động của g. Nếu

$$g'(r) = g''(r) = \ldots = g^{(\alpha-1)}(r) = 0 \neq g^{(\alpha)}(r)$$

thì tồn tại số $\delta > 0$ sao cho với mọi $x_0 \in [r - \delta, r + \delta]$, dãy $x_n = g(x_{n-1})$ hội tụ đến điểm bất động r với tốc độ hội tụ bậc α

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^\alpha} = \frac{\left|g^{(\alpha)}(r)\right|}{\alpha!}$$

Như vậy, khi sử dụng phương pháp lặp đơn, muốn có được tốc độ hội tụ cao hơn tốc độ tuyến tính như đã học, ta cần chọn hàm $g(x)$ thỏa mãn $g'(r) = 0$

Gọi r là nghiệm đơn của phương trình $f(x) = 0$. Ta viết lại phương trình dưới dạng $x = g(x)$ sao cho r là điểm cố định của $g(x)$ $\big(g(r) = r\big)$, và trong đó

$$g(x) = x - \varphi(x) f(x)$$

Dễ dàng tính được $g'(x) = 1 - \varphi(x) f'(x) - \varphi'(x) f(x)$. Giải phương trình $g'(r) = 0$, ta thu được

$$\varphi(r) = \frac{1}{f'(r)}$$

Như vậy, để có được dãy lặp hội tụ với tốc độ cao hơn, ta phải chọn hàm $\varphi(x)$ thỏa mãn điều kiện này.
Chọn $\varphi(x) = \dfrac{1}{f'(x)}$, khi đó $g(r) = r$ và $g'(r) = 0$. Phương pháp lặp đơn sẽ hội tụ với tốc độ ít nhất là bậc hai.
Thay vào ta có $g(x) = x - \dfrac{f(x)}{f'(x)}$. Đây chính là công thức của phương pháp *Newton*.
Để xác định chính xác tốc độ hội tụ, theo định lý 5, ta cần tính đạo hàm các cấp của $g(x)$ tại r cho đến khi nó bằng 0 thì dừng.
Tính đạo hàm cấp 2

$$g''(x) = \frac{f''(x)}{f'(x)} + \frac{f(x) f^{(3)}(x)}{\left[f'(x)\right]^2} - 2.\frac{f(x)\left[f''(x)\right]^2}{\left[f'(x)\right]^3}$$

Thay $x = r$ ta có $g''(r) = \dfrac{f''(r)}{2f'(r)}$. Vì trong trường hợp tổng quát, $g''(r) \neq 0$, nên phương pháp *Newton* hội tụ với tốc độ bậc hai, với hằng số tiệm cận $c = \dfrac{f''(r)}{2f'(r)}$.

6 Thuật toán

6.1 Thuật toán bằng lời

Đầu vào: Hàm $f(x)$, khoảng (a, b) và sai số ε

Đầu ra: Nghiệm xấp xỉ của phương trình $f(x) = 0$ thỏa mãn độ chính xác ở đầu vào hoặc thông báo nếu thất bại.

Bước 1. Lập trình gói tính giá trị của hàm f, f', f'' tại một điểm.

Bước 2. Lập trình gói tìm giá trị lớn nhất, nhỏ nhất của một hàm trên đoạn $[a, b]$

Bước 3. Lập trình gói xác định dấu của một số.

Bước 4. Lập trình gói kiểm tra sự xác định dấu không đổi của một hàm trên đoạn $[a, b]$

Bước 5. Kiểm tra điều kiện khoảng phân ly nghiệm $f(a)f(b) < 0$

- Nếu thỏa mãn tiếp tục bước tiếp theo
- Nếu không thỏa mãn thì thực hiện bước 11

Bước 6. Kiểm tra điều kiện hàm f', f'' xác định dấu không đổi trên $[a, b]$

- Nếu thỏa mãn tiếp tục bước tiếp theo
- Nếu không thỏa mãn thì thực hiện bước 11

Bước 7. Chọn điểm *Fourier*. Kiểm tra điều kiện $f(a)f''(a) > 0$

- Nếu thỏa mãn thì chọn $x_0 = a$
- Nếu không thỏa mãn thì chọn $x_0 = b$

Bước 8. Tìm điều kiện dừng với sai số ε ở đầu vào

- Nếu sử dụng công thức 6 thì tính
$$\epsilon = \varepsilon . \min_{x\in[a,b]} \left| f'(x) \right| \tag{9}$$
Thuật toán sẽ dừng nếu x_n thỏa mãn điều kiện
$$\left| f(x_n) \right| \le \epsilon \tag{10}$$

- Nếu sử dụng công thức 7 thì tính
$$\epsilon = \sqrt{\frac{2\min_{x\in[a,b]} \left| f'(x) \right|}{\max_{x\in[a,b]} \left| f''(x) \right|} . \varepsilon} \tag{11}$$
Thuật toán sẽ dừng nếu x_n thỏa mãn điều kiện
$$e_n = \left| x_n - x_{n-1} \right| \le \epsilon \tag{12}$$

Bước 9. Tính $x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$

(a) Sử dụng điều kiện dừng 10:
- Nếu $|f(x_1)| < \epsilon$ thì thực hiện bước tiếp theo
- Nếu $|f(x_1)| > \epsilon$ thì đặt $x_0 = x_1$ và quay lại bước này

(b) Sử dụng điều kiện dừng 12:
- Nếu $e_1 < \epsilon$ thì thực hiện bước tiếp theo
- Nếu $e_1 > \epsilon$ thì đặt $x_0 = x_1$ và quay lại bước này

Bước 10. Xuất ra nghiệm x_1.

Bước 11. Dừng chương trình. In ra nghiệm x_1 hoặc thông báo nếu thất bại.

6.2 Thuật toán bằng mã giả

Tương ứng với hai công thức sai số 6, 7, ta có hai thuật toán cho phương pháp *Newton*

Thuật toán sử dụng công thức 6

Tương ứng với công thức 6, ta có điều kiện dừng là 10. Mã giả của thuật toán:

Algorithm 1 Newton Raphson Algorithm

Input: $f(x), a, b, \varepsilon$
Output: An approximate root x_n correct to the desired degree of accuracy

procedure f, f', f''
    return
procedure MIN (g, a, b) ▷ Giá trị nhỏ nhất của g trên $[a, b]$
    $\min \leftarrow \min g(x), \quad x \in [a, b]$
    return min
procedure MAX (g, a, b) ▷ Giá trị lớn nhất của g trên $[a, b]$
    $\max \leftarrow \max g(x), \quad x \in [a, b]$
    return max
procedure SIGN (x) ▷ Dấu của số thực x
    return
procedure CHECK (g, a, b) ▷ Kiểm tra g có đổi dấu hay không
    $m \leftarrow$ sign min (g, a, b)
    $M \leftarrow$ sign max (g, a, b)
    if $m = M$ **then**
        return True
    else
        return False
procedure NEWTONRAPHSON
    $e \leftarrow$ sign $f(a), f \leftarrow$ sign $f(b)$
    if $e \neq f$ **then** ▷ Đảm bảo khoảng đang xét là "tốt"
        if check f' true **and** check f'' true **then** ▷ Điều kiện f', f'' không đổi dấu
            $c \leftarrow$ sign $f''(a)$
            if $c = e$ **then** ▷ Chọn đúng điểm Fourier
                $x_0 = a$
            else
                $x_0 = b$

$m_1 \leftarrow \min\left|f'(x)\right|$
$\epsilon \leftarrow \varepsilon.m_1$
do ▷ Vòng lặp Newton Raphson
$d \leftarrow f(x_0)/f'(x_0)$
$x_1 \leftarrow x_0 - d$
while $\left|f(x_1)\right| > \epsilon$
return x_1
else
The condition f', f'' preserve sign on $[a, b]$ is not satisfied
else
This interval doesn't have an isoltated root

Thuật toán sử dụng công thức 7

Tương ứng với công thức 7, ta có điều kiện dừng là 12. Mã giả của thuật toán:

Algorithm 2 Newton Raphson Algorithm

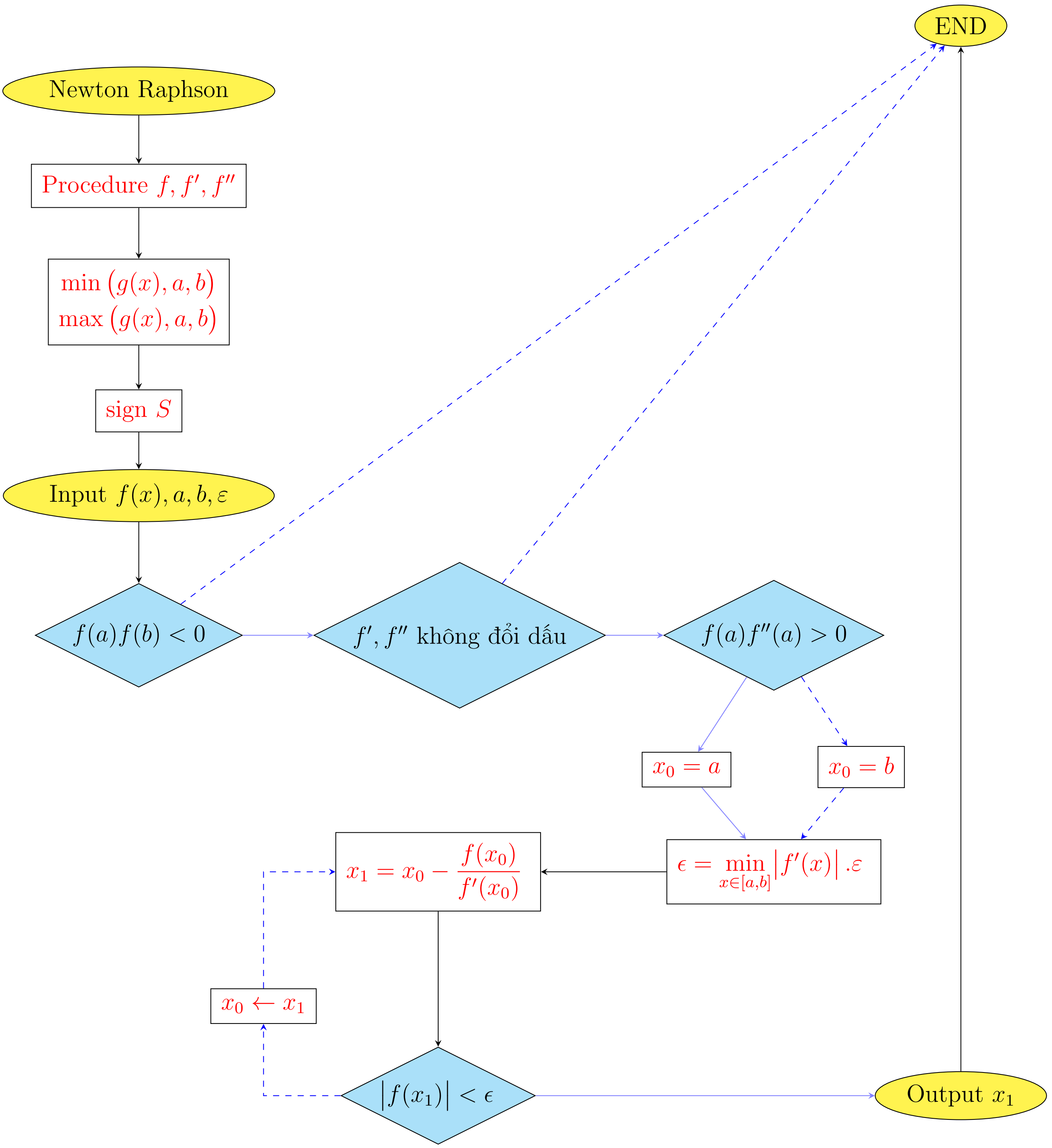
Input: $f(x), a, b, \varepsilon$
Output: An approximate root x_n correct to the desired degree of accuracy

procedure f, f', f''
return
procedure MIN (g, a, b) ▷ Giá trị nhỏ nhất của g trên $[a, b]$
$\min \leftarrow \min g(x), \quad x \in [a, b]$
return min
procedure MAX (g, a, b) ▷ Giá trị lớn nhất của g trên $[a, b]$
$\max \leftarrow \max g(x), \quad x \in [a, b]$
return max
procedure SIGN (x) ▷ Dấu của số thực x
return
procedure CHECK (g, a, b) ▷ Kiểm tra g có đổi dấu hay không
$m \leftarrow$ sign min (g, a, b)
$M \leftarrow$ sign max (g, a, b)
if $m = M$ **then**
return True
else
return False
procedure NEWTONRAPHSON
$e \leftarrow$ sign $f(a), f \leftarrow$ sign $f(b)$
if $e \neq f$ **then** ▷ Đảm bảo khoảng đang xét là "tốt"
if check f' true **and** check f'' true **then** ▷ Điều kiện f', f'' không đổi dấu
$c \leftarrow$ sign $f''(a)$
if $c = e$ **then** ▷ Chọn đúng điểm Fourier
$x_0 = a$
else
$x_0 = b$
$m_1 \leftarrow \min\left|f'(x)\right|, M_2 \leftarrow \max\left|f''(x)\right|$
$\epsilon \leftarrow \sqrt{\dfrac{2m_1}{M_2}\varepsilon}$
do ▷ Vòng lặp Newton Raphson

$d \leftarrow f(x_0)/f'(x_0)$
$x_1 \leftarrow x_0 - d$
$e_1 \leftarrow |x_1 - x_0|$
while $e_1 > \epsilon$
return x_1
else
The condition f', f'' preserve sign on $[a, b]$ is not satisfied
else
This interval doesn't have an isoltated root

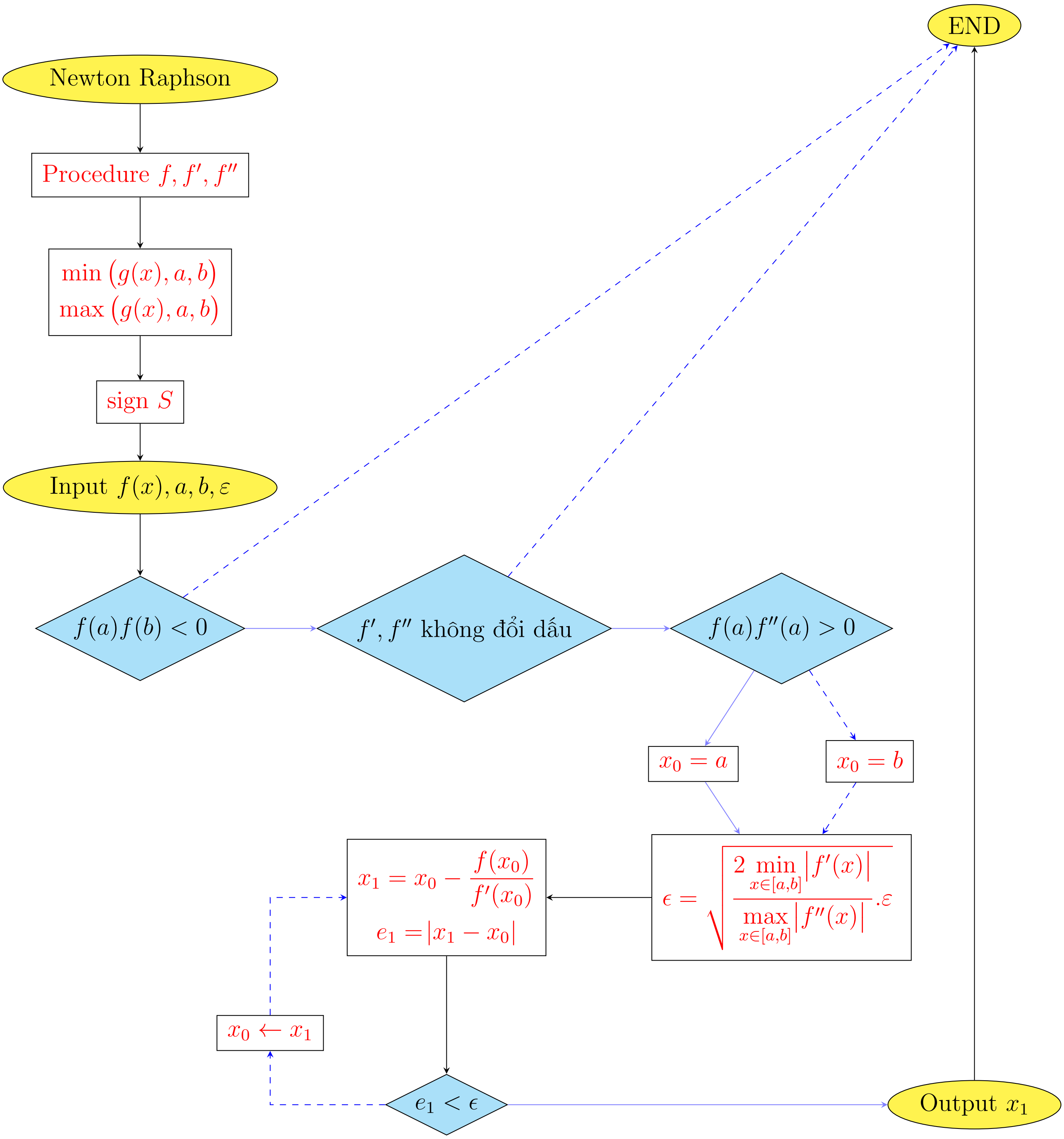
6.3 Thuật toán bằng sơ đồ khối

Tương ứng với 11 ta có sơ đồ



Hình 12: Sơ đồ khối thuật toán Newton sử dụng 11

Tương ứng với 12 ta có sơ đồ



Hình 13: Sơ đồ khối thuật toán Newton sử dụng 12

7 Một số ví dụ minh họa

7.1 Các ví dụ cơ bản

Ví dụ 7

Tính gần đúng số e với 5 chữ số đáng tin sau dấu phẩy

Ta tìm được ngay một phương trình siêu việt nhận e làm nghiệm là

$$\ln x - 1 = 0$$

Vì $f(1) < 0,\ f(5) > 0$ nên $(1, 5)$ là khoảng phân ly nghiệm của phương trình
Trên đoạn $[1; 5]$, dễ kiểm tra hàm $f(x) = \ln x - 1$ thỏa mãn các điều kiện của định lý 3 nên ta sẽ giải bài toán này bằng phương pháp Newton với hy vọng thu được nghiệm chỉ sau ít lần lặp
Yêu cầu tính gần đúng với 5 chữ số đáng tin sau dấu phẩy nghĩa là

$$\varepsilon = 0.5 \times 10^{-5} = 0.000005$$

Chương trình giải sử dụng công thức sai số giữa hai lần xấp xỉ liên tiếp và được lập trình bằng ngôn ngữ C

```c
#include<conio.h>
#include<stdio.h>
#include<math.h>

double f(double x)
{
  return log(x)-1;
}
double df(double x)
{
  double h = 1e-7;
  return((f(x + h) - f(x - h))/(2 * h));
}
double ddf(double x)
{
  float h = 1e-3;
  return((df(x + h) - df(x - h))/(2 * h));
}

//Gia tri nho nhat cua mot ham
double min(double f(double x), double a, double b)
{
  double alpha, x0 = a, e = 1e-7;
  double mini = a;

  alpha = (b - a)/10000;

  do
  {
    int loop = 10000;
    x0 = x0 + e;
    if(df(x0) > 0)
      do
      {
          x0 = x0 + alpha*df(x0);
```

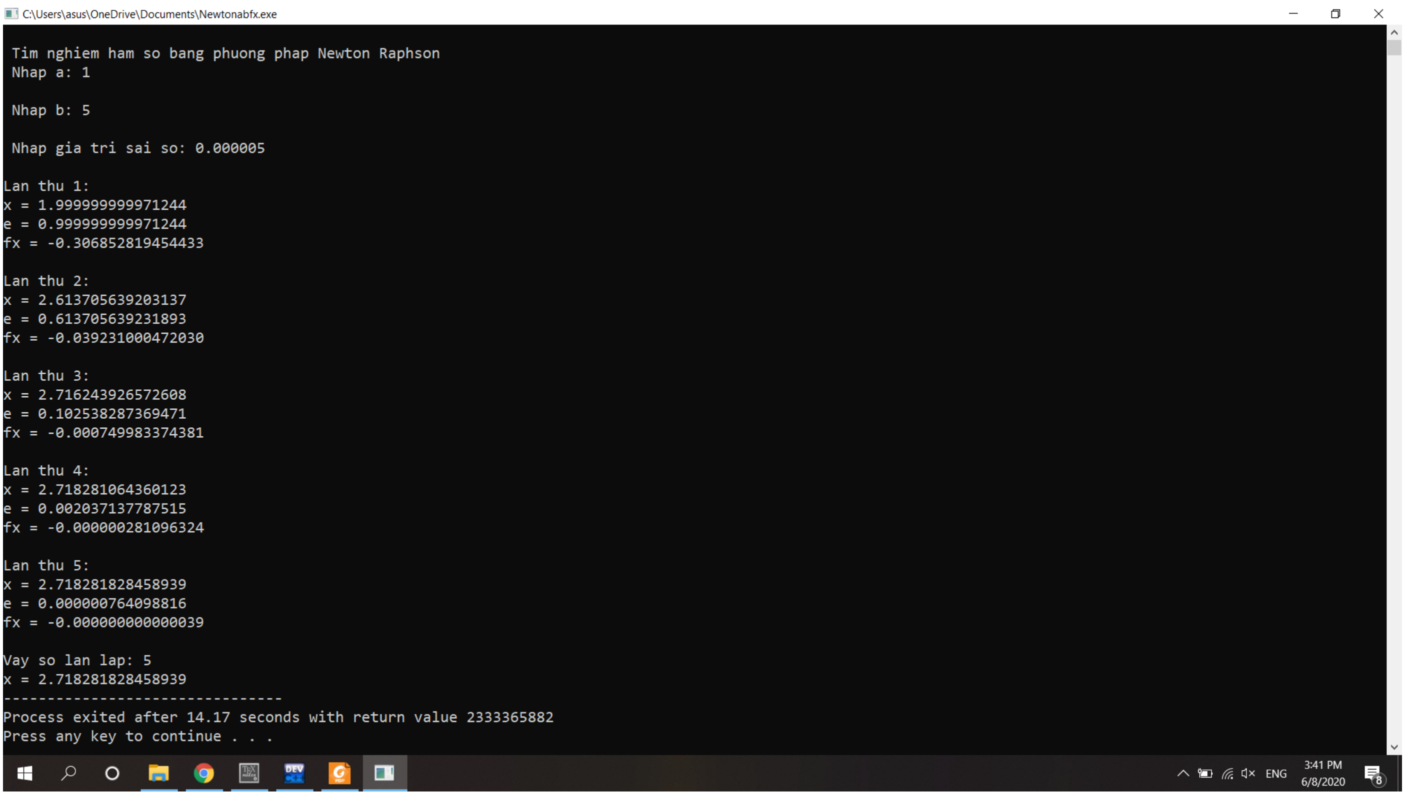
```
            loop--;
            if(loop < 0) break;
        }while(fabs(df(x0)) > e);
    else
    {
        do
        {
            x0 = x0 - alpha*df(x0);
            loop--;
            if(loop < 0) break;
        }while(fabs(df(x0)) > e);
    }
    if (x0 > b) break;
    else
    {
        if (f(x0) < f(mini))
        mini = x0;
        }
    }while(1);

    if (f(mini) < f(b)) return f(mini);
    else return f(b);
}

//Gia tri lon nhat cua mot ham
double max(double f(double x), double a, double b)
{
    double nef(double x)
    {
    return -1 * f(x);
    }
  return -1 * min(nef,a,b);
}

//Dau cua mot so
int sign(double x)
{
  if (x >= 0) return 1;
  else return -1;
}

//Kiem tra ham khong doi dau tren [a,b]
int checkf(double f(double x), double a, double b)
{
  double m1, m2;
  m1 = max(f,a,b);
  m2 = min(f,a,b);

  if (sign(m1) == sign(m2)) return 1;
  else return -1;
}

//Phuong phap Newton
double sol(double a, double b, double e)
{
  double s, x0, m1, m2, c;
  int i = 1;
  if (sign(f(a)) == sign(ddf(a))) x0 = a;
  else x0 = b;

```

```
  double f1(double x){
      return fabs(df(x));
  }
  double f2(double x){
      return fabs(ddf(x));
  }

  m1 = min(f1,a,b);
  m2 = max(f2,a,b);
  c = sqrt(2.0 * m1 * e/m2);

  //m1 = min(f1, a, b);    Su dung cong thuc sai so muc tieu
  //c = m1e;

  do
  {
    s = x0;
    x0 = x0 - f(x0)/df(x0);
    printf("\nLan thu %d:\n", i);
    printf("x = %8.15lf \n", x0);
    printf("e = %8.15lf \n", fabs(x0-s));
    printf("fx = %8.15lf \n", f(x0));
    i++;

  }while(fabs(x0-s) >= c); //|f(x0)| ≥ c    Su dung cong thuc sai so muc tieu

  if(fabs(x0-s) < c) //|f(x0)| < c     Su dung cong thuc sai so muc tieu
    {
      printf("\nVay so lan lap: %d \n", i-1);
      printf("x = %8.15lf", x0);
    }
  return(x0);
}

int main()
{
  printf("\n Tim nghiem ham so bang phuong phap Newton Raphson");
  double a, b, e;

  //Kiem tra khoang co hai dau mut trai dau
  do{
    printf("\n Nhap a: ");
    scanf("%lf", &a);
    printf("\n Nhap b: ");
    scanf("%lf", &b);

    if (sign(f(a)) == sign(f(b))) {
      printf("Day khong phai khoang cach ly nghiem! Hay nhap lai!");
    }
    if (a == b) {
      printf("a phai khac b! Hay nhap lai!");
    }
  } while ((sign(f(a)) == sign(f(b))) || (a == b));

  if (a > b)
    {
        a=a+b;
        b=a-b;
        a=a-b;
    }
```

```c

  printf("\n Nhap gia tri sai so: ");
  scanf("%lf", &e);

  //Kiem tra f',f'' khong doi dau
  int c1, c2;
  c1 = checkf(df,a,b);
  c2 = checkf(ddf,a,b);

  if ((c1 == 1) && (c2 == 1)){
    sol(a,b,e);
  }
  else{
    if (c1 != 1){
      printf("Dao ham cap mot doi dau tren doan nay. Thuat toan khong thuc hien duoc");
    }
    if (c2 != 1){
      printf("Dao ham cap hai doi dau tren doan nay. Thuat toan khong thuc hien duoc");
    }
  }
}
```

Listing 1: Chương trình thuật toán *Newton* tính gần đúng e

Màn hình kết quả



Hình 14: Chương trình thuật toán Newton tính gần đúng e

Bảng kết quả

Lần lặp	x_i	$\lvert x_i - x_{i-1}\rvert$	$f(x_i)$
1	1.99999999	0.99999999	−0.30685281
2	2.61370564	0.61370564	−0.03923100
3	2.71624393	0.10253828	−0.00074998
4	2.71828106	0.00203713	−0.00000028
5	2.71828183	0.00000076	−0.00000000

Như vậy, sau 5 lần lặp, ta thu được $x = 2.71828$ có 5 chữ số đáng tin sau dấu phẩy

Ta sẽ phân tích chương trình 1 này

Kiểm tra khoảng (a, b)

Sử dụng định lý 3, ta chỉ chấp nhận người dùng nhập vào khoảng (a, b) thỏa mãn

- Nếu người dùng nhập $a = b$ phải yêu cầu nhập lại.
- Nếu người dùng nhập $a > b$ phải lập trình hoán vị 2 số a, b.

```c
//Kiem tra khoang co hai dau mut trai dau
  do{
    printf("\n Nhap a: ");
    scanf("%lf", &a);
    printf("\n Nhap b: ");
    scanf("%lf", &b);

    if (sign(f(a)) == sign(f(b))) {
      printf("Day khong phai khoang cach ly nghiem! Hay nhap lai!");
    }
    if (a == b) {
      printf("a phai khac b! Hay nhap lai!");
    }
  } while ((sign(f(a)) == sign(f(b))) || (a == b));

  if (a > b)
    {
        a=a+b;
        b=a-b;
        a=a-b;
    }
```

Listing 2: Kiểm tra khoảng đầu vào $(a; b)$

- Hàm $f(x)$ xác định tại $x = a$ và $x = b$. Đây cũng là một sơ hở trong khâu kiểm tra đầu vào, ta không kiểm tra được tính liên tục của f, f', f'' trên $[a, b]$ bằng máy tính. Việc kiểm tra tính liên tục ta thực hiện thủ công ở ngoài.
- Khoảng (a, b) là khoảng phân ly nghiệm của f. Có nghĩa là f không có nghiệm nào khác ngoài r. Điều này được đảm bảo bởi điều kiện f' không đổi dấu trên $[a, b]$
- Khẳng định trên rất quan trọng, vì nếu trong khoảng này, f còn nghiệm nào khác, ta không thể kiểm soát được phương pháp sẽ hội tụ về nghiệm nào. Ta rất có khả năng rơi vào tình huống "nhảy nghiệm".
- Hơn thế nữa, ta giả sử $f(a)f(b) < 0$ và sử dụng 4 trường hợp cơ bản như trong hình 8, có nghĩa là bỏ qua các trường hợp nghiệm bội chẵn của f (nếu có).

- Chỉ kiểm tra điều kiện $f(a)f(b) < 0$, ta không thể biết được r là nghiệm đơn hay nghiệm bội lẻ, nhưng ta có thể cảm nhận được điều này khi nhìn vào kết quả. Một số kết quả về trường hợp nghiệm bội sẽ được bàn đến sau.

Kiểm tra điều kiện f', f'' giữ dấu không đổi trên (a, b)

Báo cáo này sử dụng phương pháp Steapest Descent để tìm giá trị lớn nhất, nhỏ nhất của một hàm trên đoạn đóng $[a, b]$. Code của chương trình ở 13.

Ý tưởng tổng quát

Giả sử ta đang muốn tìm giá trị nhỏ nhất của hàm $f(x)$ trên đoạn đóng $[a, b]$, gọi x^* là điểm cực tiểu địa phương của f. Ý tưởng tổng quát của phương pháp dựa trên vòng lặp 2 bước sau:

Bước 1. Giả sử x_n là điểm ta tìm được sau vòng lặp thứ n. Tại x_n, ta chọn hướng d_n sao cho hàm $f(x)$ giảm khi x ra xa x_n theo hướng d_n.

Bước 2. Đặt $x_{n+1} = x_n + \eta d_n$, với η được chọn sao cho hàm $\varphi(\eta)$ nhỏ nhất có thể

$$\varphi(\eta) = f(x_{n+1}) = f(x_n + \eta d_n),$$

η được gọi là *learning rate*.

Trong phương pháp Steapest Descent, ta chọn $d_n = -f(x_n)$, là hướng mà hàm giảm nhanh nhất khi ra xa x_n.
Theo ý tưởng này, ta thấy rằng hàm f không đổi dấu trên (a, b) khi và chỉ khi

$$\min_{[a,b]} f \quad \max_{[a,b]} f \quad > 0$$

```
//Kiem tra ham khong doi dau tren [a,b]
int checkf(double f(double x), double a, double b)
{
  double m1, m2;
  m1 = max(f,a,b);
  m2 = min(f,a,b);

  if (sign(m1) == sign(m2)) return 1;
  else return -1;
}

//Kiem tra f',f'' khong doi dau
  int c1, c2;
  c1 = checkf(df,a,b);
  c2 = checkf(ddf,a,b);

  if ((c1 == 1) && (c2 == 1)){
    sol(a,b,e);
  }
  else{
    if (c1 != 1){
      printf("Dao ham cap mot doi dau tren doan nay. Thuat toan khong thuc hien duoc");
    }
    if (c2 != 1){
      printf("Dao ham cap hai doi dau tren doan nay. Thuat toan khong thuc hien duoc");
    }
```

```
}
```

Listing 3: Kiểm tra điều kiện đạo hàm 2 cấp không đổi dấu

Phương pháp tính đạo hàm

1. Như ta đã biết, phương pháp *Newton* có được tốc độ hội tụ bậc hai là vì nó có sự hiệu chính liên tục, nhờ vào đường tiếp tuyến, để đảm bảo hướng đi không bị sai lệch. Nhưng đổi lại, ở mỗi lần lặp, khối lượng tính toán lớn và không thuận tiện khi lập trình, vì với mỗi phương trình ta lại phải nhập f, f' và f''.

2. Để khắc phục điều này, ta tính đạo hàm dựa vào định nghĩa

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0 - h)}{2h}$$

```
double df(double x)
{
  double h = 1e-7;
  return((f(x + h) - f(x - h))/(2 * h));
}
double ddf(double x)
{
  float h = 1e-3;
  return((df(x + h) - df(x - h))/(2 * h));
}
```

Listing 4: Xấp xỉ đạo hàm theo định nghĩa

Cách làm này làm giảm khối lượng tính toán và thuận tiện hơn khi lập trình.
Tuy nhiên, cần lưu ý rằng, không được chọn h quá nhỏ nếu không ta sẽ tính ra f' rất lớn, cũng không được chọn h quá lớn để đảm bảo xấp xỉ là "chấp nhận được"

> **1.** Trong thực tế, đôi khi người ta cũng rút gọn khối lượng tính toán đạo hàm tại mỗi bước bằng cách xấp xỉ $f'(x_n)$ bởi $f'(x_0)$ và sử dụng công thức lặp
>
> $$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_0)}$$
>
> Tuy nhiên, cần phải suy tính kĩ để cân bằng giữa tốc độ hội tụ với khối lượng tính toán, vì công thức lặp này chỉ hội tụ tuyến tính.
> **2.** Phương pháp này có thể sử dụng trong trường hợp f' thay đổi không quá nhiều.

Định lý 11

Dãy lặp theo công thức $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_0)}$ hội tụ với tốc độ tuyến tính

Chứng minh. Gọi r là nghiệm của phương trình $f(x) = 0$. Khi đó $f(r) = 0$ và $e_n = x_n - r$. Từ công thức lặp suy ra

$$e_{n+1} = e_n - \frac{f(e_n + r)}{f'(x_0)} = e_n - \frac{f(r) + e_n f'(r) + \cdots}{f'(x_0)} = e_n \left(1 - \frac{f'(r)}{f'(x_0)}\right) + o(e_n^2)$$

Bỏ qua e_n^2 và các vô cùng bé bậc cao hơn của e_n^2 và đặt $A = 1 - \dfrac{f'(r)}{f'(x_0)}$, biểu diễn trên trở thành

$$e_{n+1} = Ae_n$$

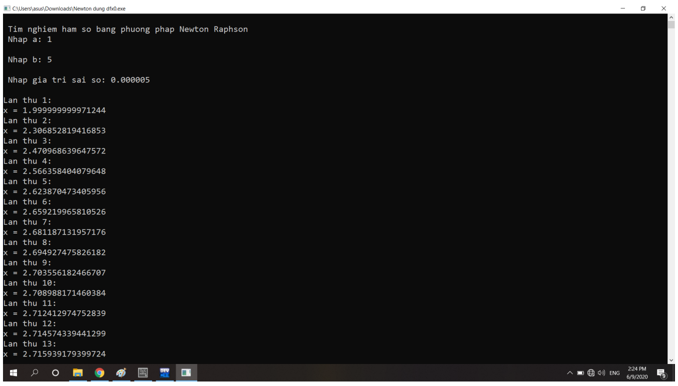
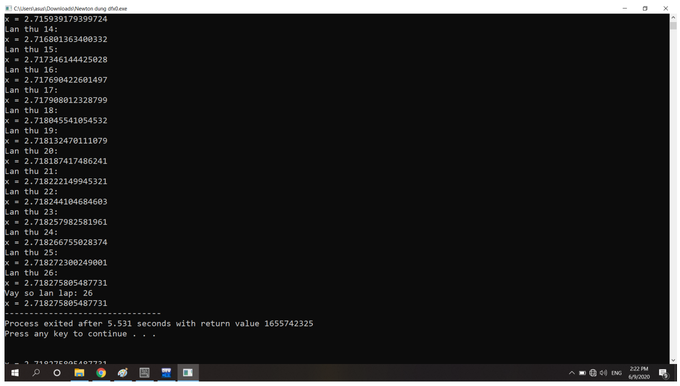
Điều này chứng tỏ rằng công thức lặp hội tụ tuyến tính. □

Chương trình giải thay đổi như sau:

```
double sol(double a, double b, double e)
{
  double s, x0, dfx0;
  int i=1;
  if (sign(f(a)) == sign(ddf(a))) x0=a;
  else x0=b;
  dfx0 = df(x0); //Co dinh f'(x0)
  do
  {
    s = x0;
    x0 = x0 - f(x0)/dfx0;
    printf("\nLan thu %d:\n", i);
    printf("x = %8.15lf", x0);
    i++;
  }while(fabs(x0-s) >= e); //Dieu kien dung |x_n - x_{n-1}| < ε

  if(fabs(x0-s) < e)
    {
      printf("\nVay so lan lap: %d \n", i-1);
      printf("x = %8.15lf", x0);
    }
  return(x0);
}
```

Listing 5: Phương pháp "tựa" Newton xấp xỉ đạo hàm bởi $f'(x_0)$

Màn hình kết quả

Hình 15: Phương pháp "tựa" Newton xấp xỉ đạo hàm bởi $f'(x_0)$

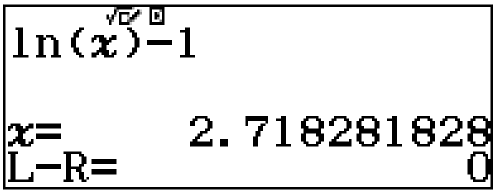
Như vậy, chỉ xấp xỉ $f'(x_n)$ bởi $f'(x_0)$ và thay đổi công thức lặp, tốc độ hội tụ tăng lên đáng kể. Lấy $\varepsilon = 0.5 \times 10^{-5} = 0.000005$, ta phải cần đến 27 lần lặp

Chú ý: Ở đây ta sử dụng điều kiện dừng $|x_{n+1} - x_n| < \varepsilon$

Kiểm tra tính đúng đắn của chương trình

Một số cách có thể sử dụng để kiểm tra tính đúng đắn của chương trình

1. Ưu tiên hàng đầu là máy tính cầm tay *CASIO*. Đây là công cụ hỗ trợ phổ biến, có thể sử dụng bất cứ lúc nào. Dòng máy *CASIO fx-570 VN PLUS* hiện nay cho kết quả chính xác đến 8 chữ số thập phân
   Ta có thể sử dụng chức năng Solve để tìm nghiệm của phương trình



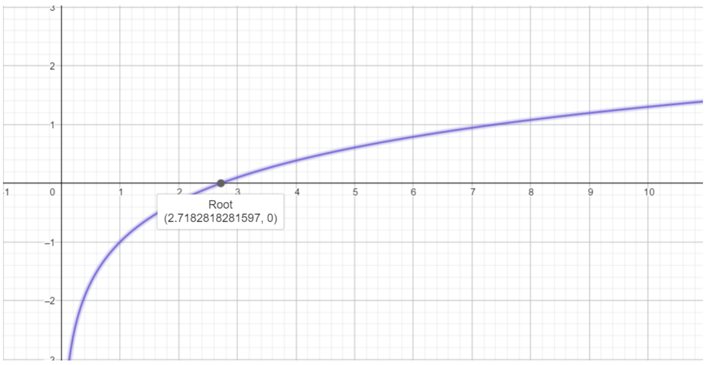
Hình 16: Giải phương trình $\ln x - 1 = 0$ bằng chức năng Solve

   Hoặc đơn giản hơn, thay ngược lại nghiệm của phương trình để xem kết quả có đúng không

Hình 17: Thay ngược lại vào phương trình bằng Casio

   Như vậy, kết quả chạy chương trình là chính xác.

2. Công cụ phổ biến tiếp theo có thể kể đến là các ứng dụng vẽ đồ thị và cho phép xác định tọa độ giao điểm như *GEOGEBRA, CABRI,*... Ứng dụng *GEOGEBRA* phiên bản mới nhất có thể cài đặt trực tiếp trên máy tính, không cần sử dụng *Internet* và hiển thị giao điểm đối tượng với 13 chữ số thập phân. Tuy nhiên có một hạn chế là không phải hàm nào cũng có thể vẽ được đồ thị.



Hình 18: Vẽ đồ thị hàm $f(x) = \ln x - 1$ bằng Geogebra

3. Trang web *Wolfram Alpha* cũng là một lựa chọn không tồi. Các chức năng miễn phí trên web đủ để người dùng giải phương trình, tìm cực trị, vẽ đồ thị,... Tuy nhiên đòi hỏi phải có kết nối *Internet*, và kết quả chỉ hiển thị cỡ khoảng $6 - 7$ chữ số thập phân. Địa chỉ web: https://www.wolframalpha.com/

4. Các hàm cài đặt trong *Matlab* và các thư viện trong *Python* cũng rất đáng tin cậy và cho kết quả nhanh chóng. Tuy nhiên đòi hỏi phải hiểu biết ngôn ngữ và thời gian lập trình chương trình cho ra kết quả. Các thư viện Python thông dụng là *numpy, scipy, sympy, ...*

5. Thư viện *GSL - GNU Scientific Library* là thư viện mở rộng trên ngôn ngữ lập trình $C, C++$. Nó là sản phẩm có bản quyền mã nguồn mở, bao gồm nhiều hàm đã được lập trình sẵn để hỗ trợ việc giải số trên $C, C++$ với các hàm như giải phương trình, giải hệ phương trình, trị riêng,... Tuy nhiên quá trình tải về thư viện khá lâu, và đòi hỏi người xem phải thành thạo C mới có thể hiểu mã nguồn và thay tham số. Địa chỉ trang web: https://www.gnu.org/software/gsl/

Phương pháp hội tụ bậc hai

Trong ví dụ này ta sử dụng công thức sai số 7. Quan sát bảng kết quả dưới đây

Lần lặp	x_i
1	1.99999999
2	2.61370564
3	2.71624393
4	2.71828106
5	2.71828182

Ta thấy sau mỗi lần lặp, số chữ số chính xác tăng lên gần như 2 lần: $0, 1, 3, 7, \ldots$.
Đây chính là ý nghĩa của tốc độ hội tụ bậc hai.

So sánh giữa hai công thức sai số

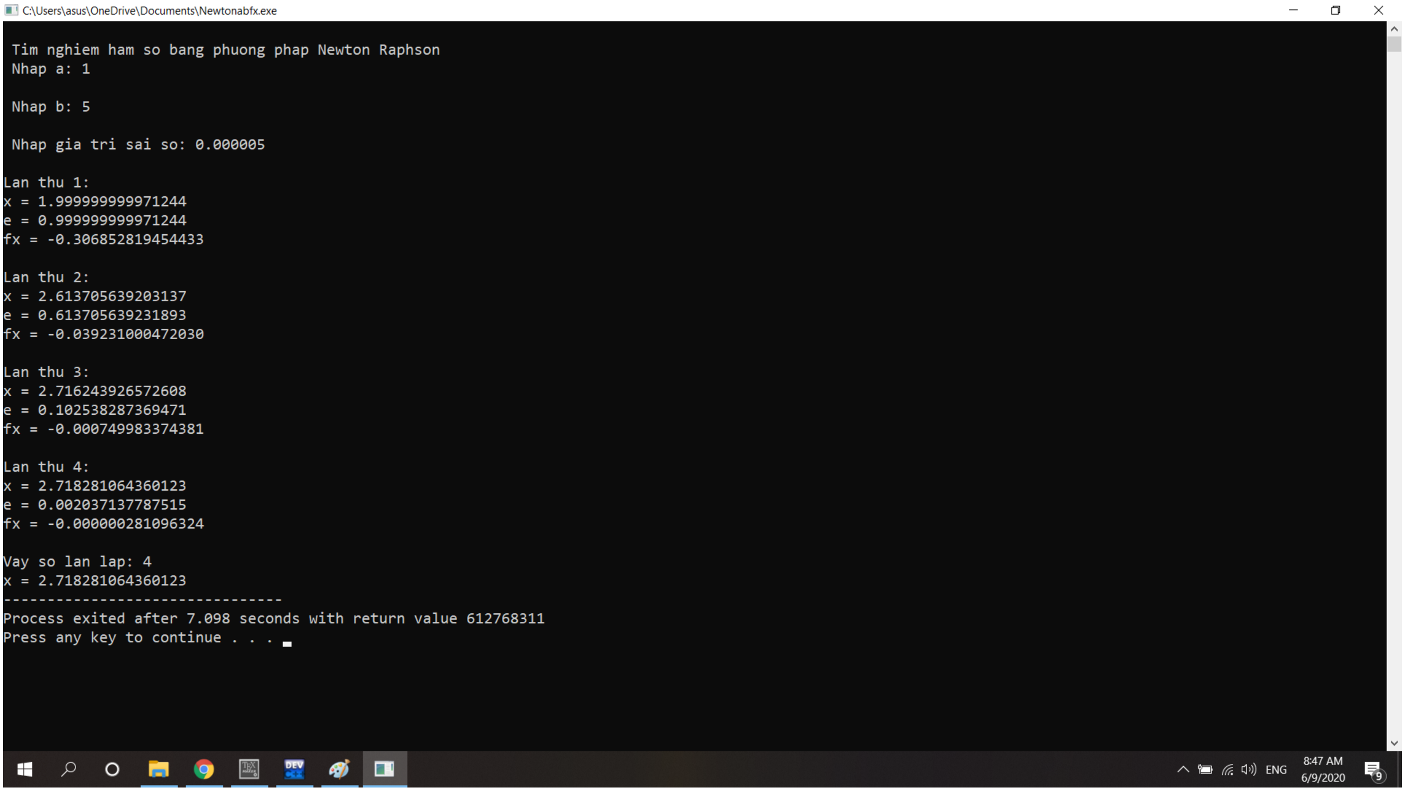
Trong code ta chỉ cần thay đổi công thức tính ϵ và điều kiện dừng của thuật toán như sau:

```
double sol(double a, double b, double e)
{
  double s, x0, m1, m2, c;
  int i = 1;
  if (sign(f(a)) == sign(ddf(a))) x0 = a;
  else x0 = b;

  double f1(double x){
      return fabs(df(x));
  }
  double f2(double x){
      return fabs(ddf(x));
  }

  m1 = min(f1,a,b);
  c = m1 * e; //Cong thuc tinh sai so ϵ = m1ε

  do
  {
    s = x0;
    x0 = x0 - f(x0)/df(x0);
    printf("\nLan thu %d:\n", i);
    printf("x = %8.15lf \n", x0);
    printf("e = %8.15lf \n", fabs(x0-s));
    printf("fx = %8.15lf \n", f(x0));
    i++;

  }while(fabs(f(x0)) >= c ); //Dieu kien dung |f(xn)| < ϵ

  if(fabs(f(x0)) < c)
    {
      printf("\nVay so lan lap: %d \n", i-1);
      printf("x = %8.15lf", x0);
    }
  return(x0);
```

```
}
```

Listing 6: Tính gần đúng e sử dụng công thức sai số mục tiêu

Màn hình kết quả



Hình 19: Tính gần đúng e với điều kiện dừng $\left|f(x_n)\right| < m_1\varepsilon$

Bảng kết quả

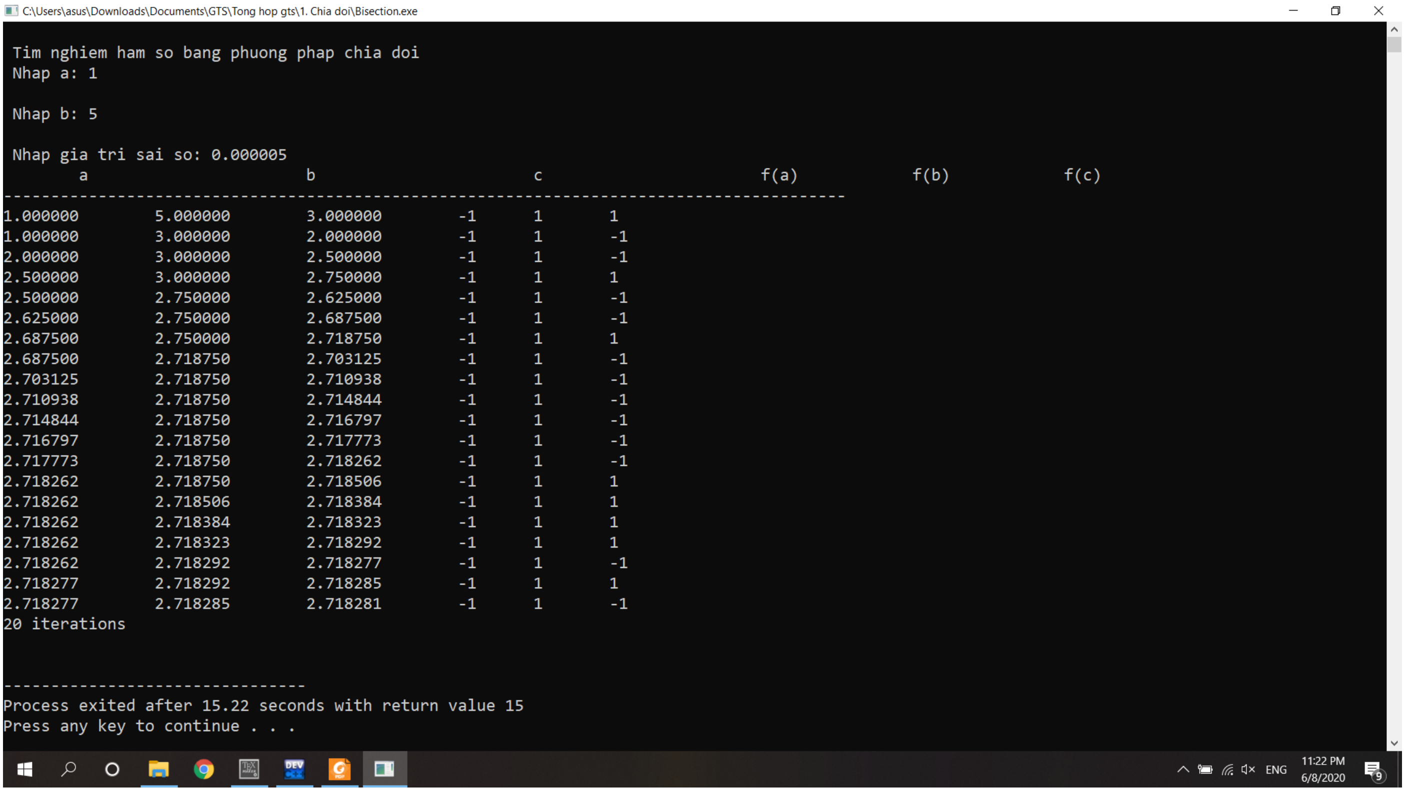
Lần lặp	x_i	$\lvert x_i - x_{i-1}\rvert$	$f(x_i)$
1	1.99999999	0.99999999	-0.30685281
2	2.61370564	0.61370564	-0.03923100
3	2.71624393	0.10253828	-0.00074998
4	2.71828106	0.00203713	-0.00000028

Như vậy, với bài toán này, sử dụng công thức sai số mục tiêu cần 4 lần lặp và sử dụng công thức sai số giữa hai 2 lần xấp xỉ liên tiếp cần 5 lần lặp.

So sánh với phương pháp chia đôi

Code chương trình sử dụng phương pháp chia đôi giải ví dụ trên ở 12. Ở đây ta sử dụng điều kiện dừng $|b_n - a_n| \leq \varepsilon$, với $\varepsilon = 0.5 \times 10^{-5} = 0.000005$

Màn hình kết quả



Hình 20: Giải phương trình $\ln x - 1 = 0$ bằng phương pháp chia đôi

Bảng kết quả

Lần lặp	a	b	c	$f(a)$	$f(b)$	$f(c)$
1	1.000000	5.000000	3.000000	$-$	$+$	$+$
2	1.000000	3.000000	2.000000	$-$	$+$	$-$
3	2.000000	3.000000	2.500000	$-$	$+$	$-$
4	2.500000	2.750000	2.625000	$-$	$+$	$+$
5	2.500000	2.750000	2.625000	$-$	$+$	$-$
6	2.625000	2.750000	2.687500	$-$	$+$	$-$
7	2.687500	2.750000	2.718750	$-$	$+$	$+$
8	2.687500	2.718750	2.703125	$-$	$+$	$-$
9	2.703125	2.718750	2.710938	$-$	$+$	$-$
10	2.710938	2.718750	2.714844	$-$	$+$	$-$
11	2.714844	2.718750	2.716797	$-$	$+$	$-$
12	2.716797	2.718750	2.717773	$-$	$+$	$-$
13	2.717773	2.718750	2.718282	$-$	$+$	$-$
14	2.718262	2.718750	2.718506	$-$	$+$	$+$
15	2.718262	2.718506	2.718384	$-$	$+$	$+$
16	2.718262	2.718384	2.718323	$-$	$+$	$+$
17	2.718262	2.718323	2.718292	$-$	$+$	$+$
18	2.718262	2.718292	2.718277	$-$	$+$	$-$
19	2.718277	2.718292	2.718285	$-$	$+$	$+$
20	2.718277	2.718285	2.718281	$-$	$+$	$-$

Từ kết quả trên ta rút ra nhận xét

Tiêu chí \ Phương pháp	**Chia đôi**	**Tiếp tuyến**
Điều kiện áp dụng	Chỉ cần khoảng phân ly	Chặt chẽ hơn: Khoảng phân ly, f' và f'' không đổi dấu
Số lần lặp	Nghiệm thỏa mãn độ chính xác sau 20 lần	Hội tụ nhanh hơn nhiều, chỉ cần 5 lần
Khối lượng tính toán ở mỗi lần lặp	Tìm trung điểm c_n của $[a_n, b_n]$ và so sánh dấu của f tại 3 điểm	Cần tính giá trị $f(x_n)$ và $f'(x_n)$
Tính chất dãy lặp	Đoạn $[a_n, b_n]$ thu hẹp dần nhưng luôn chứa nghiệm đúng r	Dãy lặp $\{x_n\}$ đơn điệu tăng và tiến dần về r

Trong các ví dụ dưới đây ta đều sử dụng công thức sai số giữa 2 lần xấp xỉ liên tiếp

Ví dụ 8

Biết phương trình

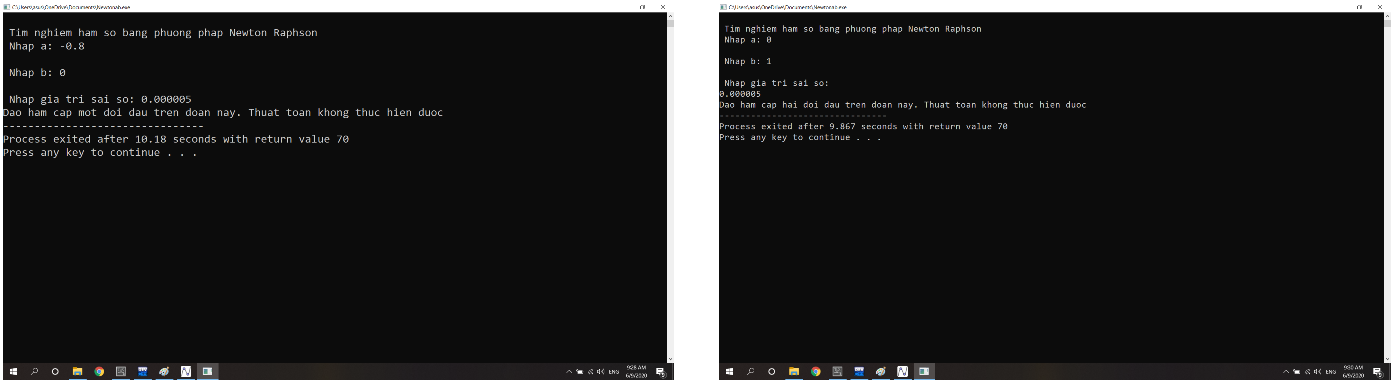
$$2e^{-x} = \frac{1}{x+2} + \frac{1}{x+1}$$

có 2 nghiệm lớn hơn -1. Tìm các nghiệm này với 5 chữ số đáng tin

Xét $f(x) = e^{-x} - \dfrac{1}{x+2} - \dfrac{1}{x+1}$, ta tính được

$$f(-0.8) = -1.38; \ \ f(0) = 0.5; \ \ f(1) = -0.0976$$

Mà phương trình có 2 nghiệm lớn hơn -1 suy ra $(-0.8; 0)$ và $(0; 1)$ là hai khoảng phân ly nghiệm của phương trình.
Để đạt được 5 chữ số đáng tin thì ta phải có $\varepsilon = 0.5 \times 10^{-5} = 0.000005$. Chạy chương trình



Hình 21: Chương trình chạy ví dụ 8

Ta xuất ra thông báo cho người dùng biết hàm f không thỏa mãn các điều kiện của định lý 3 trên khoảng $(-0.8; 0)$.
Xét tiếp khoảng $(0; 1)$. Tương tự ta cũng có $\varepsilon = 0.5 \times 10^{-5} = 0.000005$.
Như vậy, không phải lúc nào cũng có thể sử dụng được phương pháp *Newton*. Ta cần kiểm tra nếu hàm $f(x)$ thỏa mãn tất cả các điều kiện của định lý 3 thì mới áp dụng.

Ví dụ 9

Tìm một nghiệm dương của phương trình

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6}e^{0.3x}$$

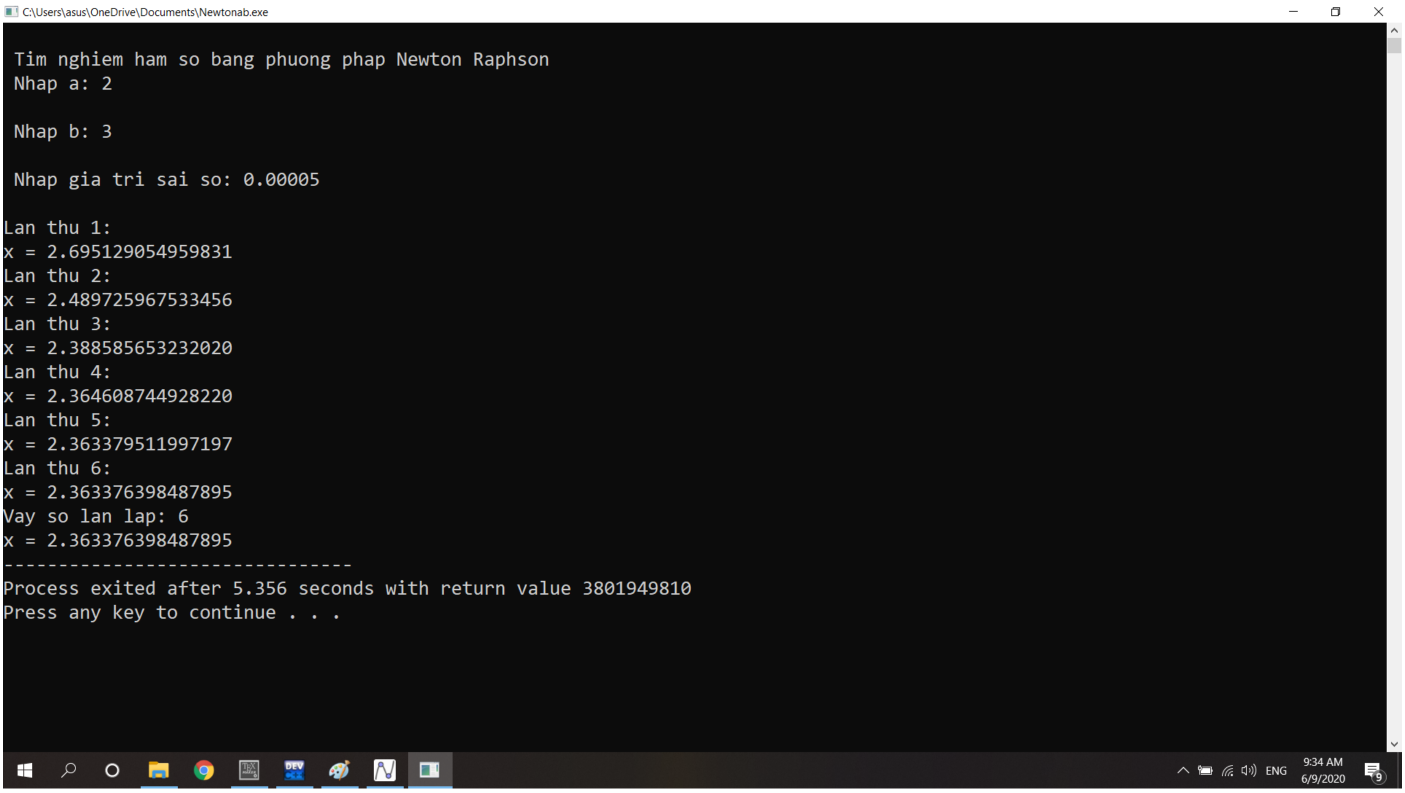
với 5 chữ số đáng tin

Đặt $e^x - 1 - x - \frac{x^2}{2} - \frac{x^3}{6}e^{0.3x}$, ta tính được

$$f(0) = 0;\ f(1) = -0.0067;\ f(2) = -0.0404;\ f(3) = 0.5173$$

Suy ra f có một nghiệm dương trong khoảng $(2; 3)$. Để nghiệm tìm được có 5 chữ số đáng tin thì sai số đầu vào là

$$\varepsilon = 0.5 \times 10^{-4} = 0.00005$$



Hình 22: Chương trình chạy ví dụ 9

Kết quả tìm được là $x = 2.3634$ sau 6 lần lặp.

Ví dụ 10

Phương trình $f(x) = 0$, với

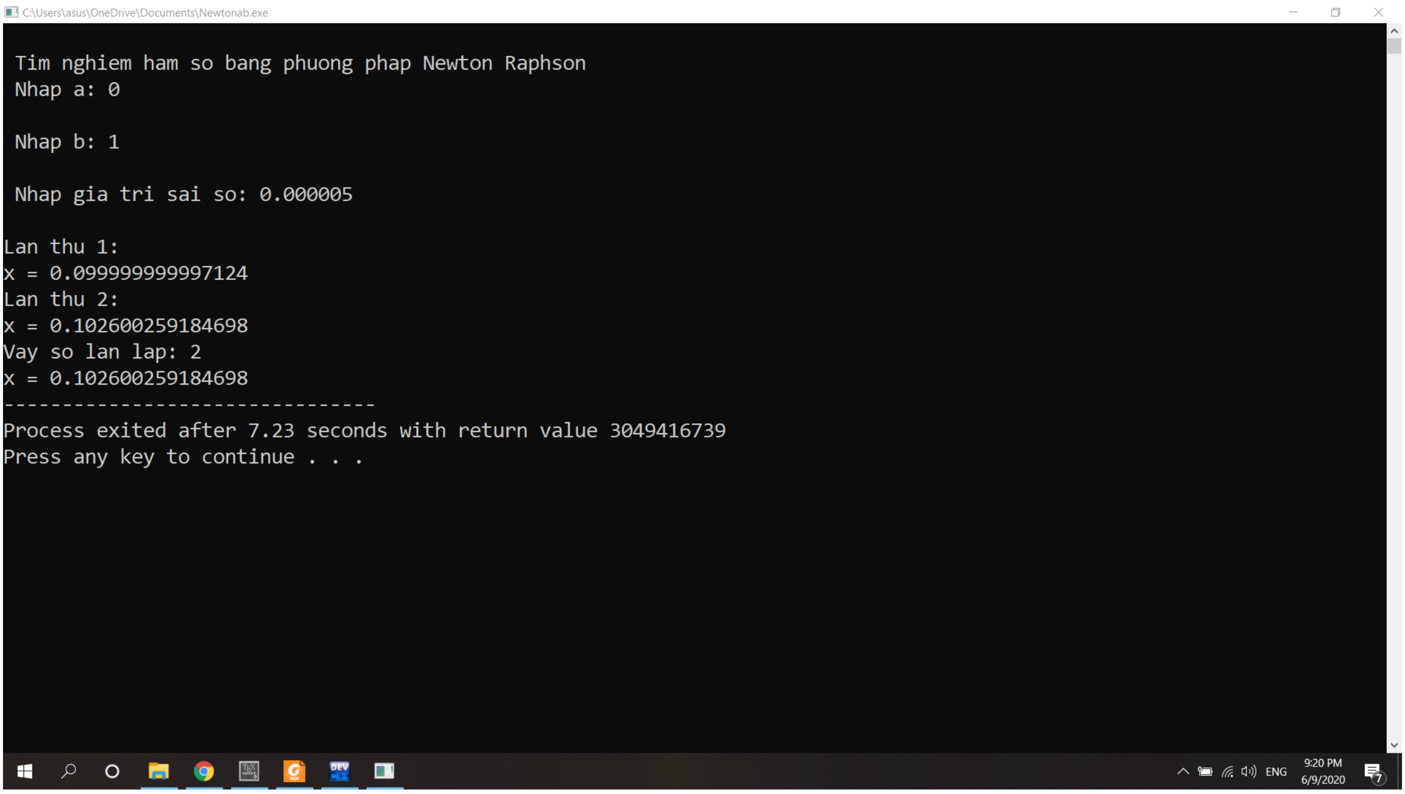
$$f(x) = 0.1 - x + \frac{x^2}{(2!)^2} - \frac{x^3}{(3!)^2} + \frac{x^4}{(4!)^2} - \cdots$$

có 1 nghiệm trong khoảng $(0, 1)$. Tìm nghiệm này với 5 chữ số đáng tin

Do sai số $\varepsilon = 0.5 \times 10^{-5} = 0.000005$ và $\dfrac{x^6}{(6!)^2} < \dfrac{1}{(6!)^2} < \varepsilon$ nên ta có thể qua các số hạng trong chuỗi từ mũ 6. Tuy nhiên khi cộng trừ nó vẫn có thể ảnh hưởng đến chữ số thứ 5 nên để chắc chắn, ta bỏ qua các số hạng có số mũ lớn hơn hoặc bằng 7. Xét

$$g(x) = 0.1 - x + \frac{x^2}{(2!)^2} - \frac{x^3}{(3!)^2} + \frac{x^4}{(4!)^2} - \frac{x^5}{(5!)^2} + \frac{x^6}{(6!)^2}$$

Kết quả chạy chương trình như sau



Hình 23: Chương trình chạy ví dụ 10

Như vậy ta thu được nghiệm $x \simeq 0.10260$

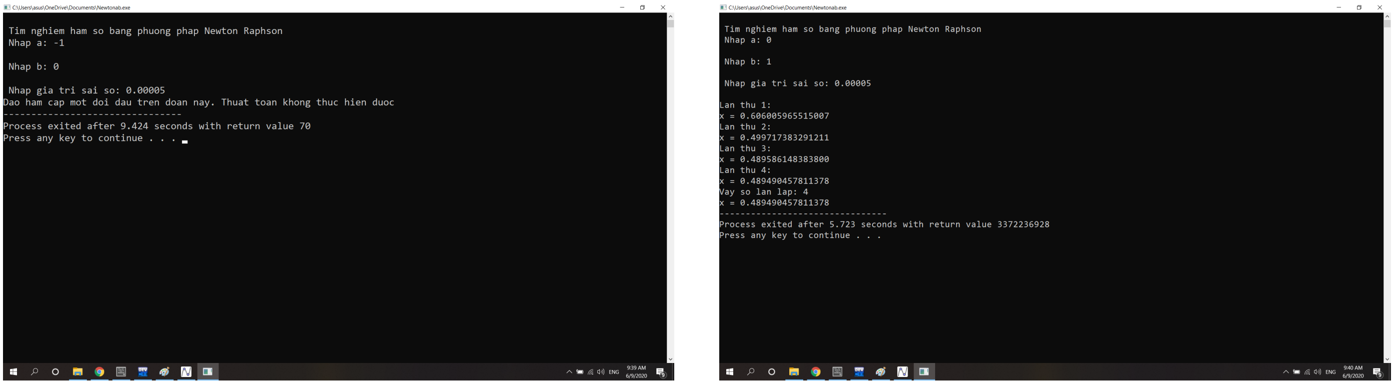
Ví dụ 11

Chứng minh rằng phương trình

$$f(x) = \cos\left(\frac{\pi x + \pi}{8}\right) + 0.148x - 0.9062 = 0$$

có một nghiệm trong khoảng $(-1, 0)$ và một nghiệm trong khoảng $(0, 1)$. Tìm các nghiệm này với 4 chữ số đáng tin

Tính toán trực tiếp $f(-1) = -0.0542$; $f(0) = 0.0177$; $f(1) = -0.0511$ nên phương trình có một nghiệm trong khoảng $(-1, 0)$ và một nghiệm trong khoảng $(0, 1)$.
Sai số đầu vào để đạt được 4 chữ số đáng tin đối với cả 2 khoảng này là $\varepsilon = 0.5 \times 10^{-4} = 0.00005$
Trên $(-1; 0)$ hàm f có đạo hàm cấp 1 đổi dấu. Ta không thể tìm nghiệm trong khoảng này bằng phương pháp *Newton*



Hình 24: Chương trình chạy ví dụ 11

Trên khoảng $(0; 1)$ ta tìm được nghiệm $x = 0.4895$ sau 4 lần lặp. Như vậy, mặc dù phương trình có 2 nghiệm nhưng trong trường hợp này ta chỉ tìm được 1 nghiệm bằng phương pháp *Newton*

1. Trong chương trình ta cần bổ sung thêm dòng sau

```
#define PI 3.14159265358979323846
```

2. Ngôn ngữ lập trình C không có sẵn hằng số π cho ta dùng nên cần tự định nghĩa π như trên. Từ đó, một cách tự nhiên, ta có thể phát biểu bài toán
Tính gần đúng hằng số π với số chữ số đáng tin tùy ý theo yêu cầu
Bài toán này tương tự như ví dụ 7, ta có thể lựa chọn một hàm siêu việt thích hợp nhận π làm nghiệm mà thỏa mãn các điều kiện của định lý 3 để sử dụng phương pháp *Newton*

Ví dụ 12

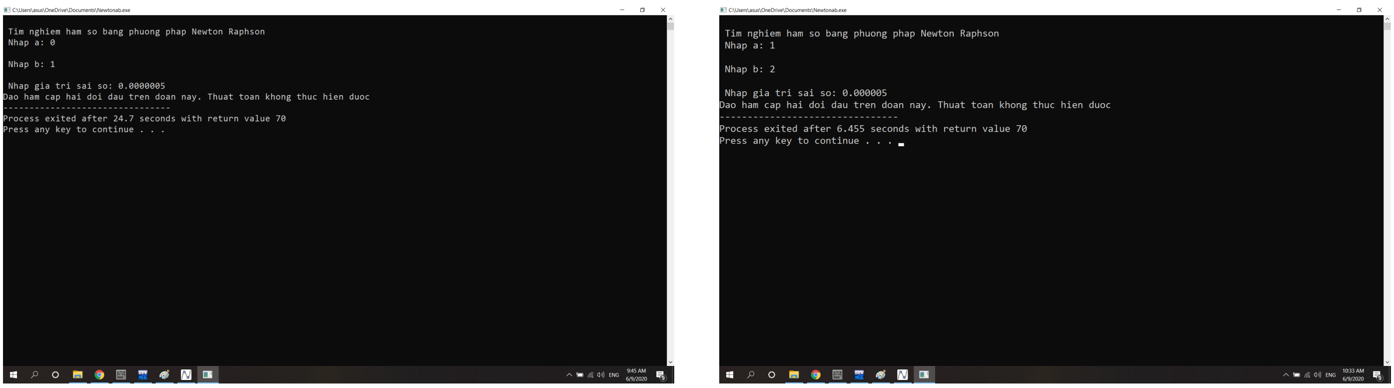
Tìm tất cả các nghiệm dương của phương trình

$$10 \int_0^x e^{-x^2}\, dt = 1$$

với 6 chữ số đáng tin

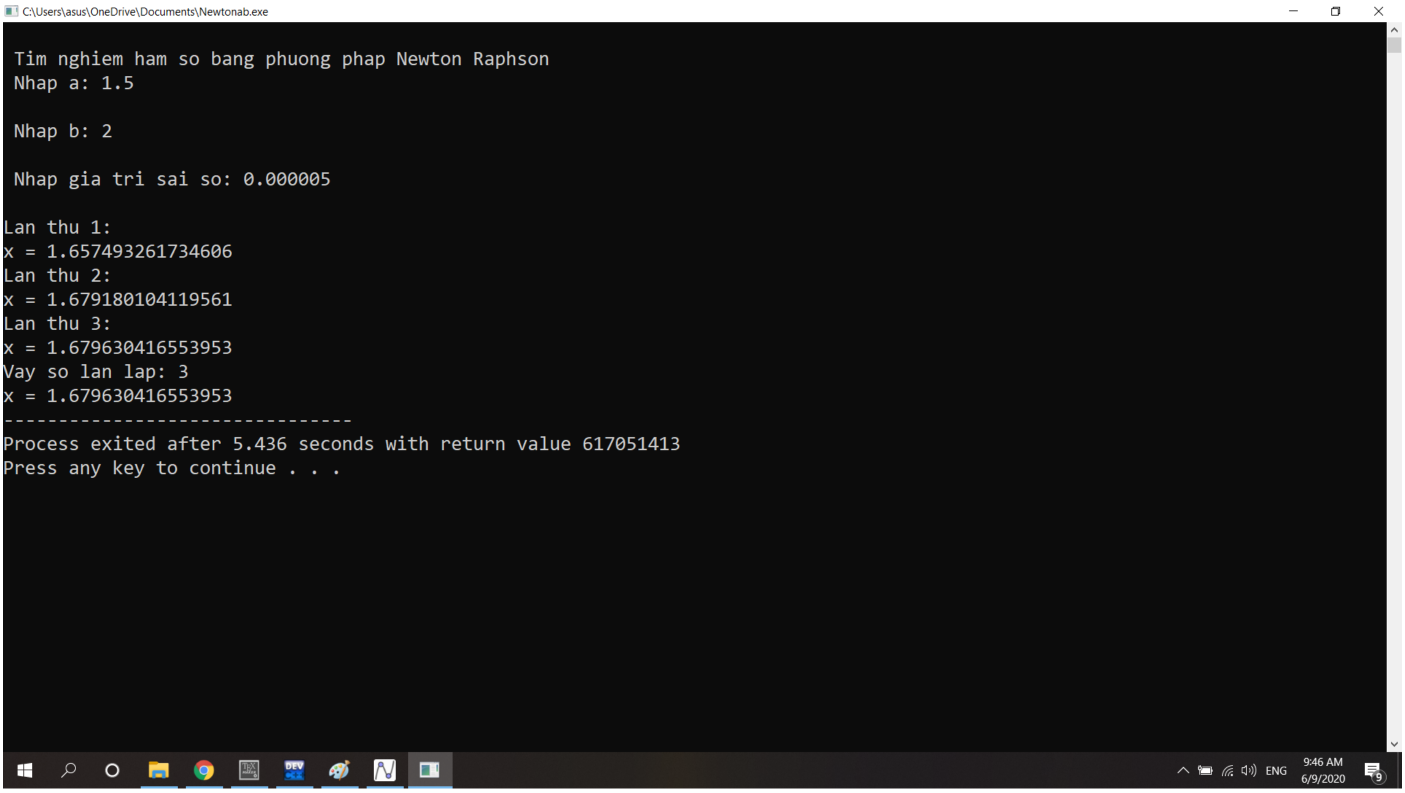
Ta viết lại phương trình thành $f(x) = 10xe^{-x^2} - 1 = 0$. Tính được

$$f(0) = -1; \;\; f(1) = 2.6788; \;\; f(2) = -0.6337$$



Hình 25: Ví dụ 12 hàm không thỏa mãn các điều kiện trong định lý 3

Nên $(0;1)$ và $(1;2)$ là 2 khoảng phân ly nghiệm dương của phương trình
Mặt khác, dễ chứng minh được $f(x) < 0$ với mọi $x > 2$ nên f không còn khoảng phân ly nghiệm dương nào khác.
Trên khoảng $(0;1)$, ta lấy sai số đầu vào là $\varepsilon = 0.5 \times 10^{-6} = 0.0000005$. Chạy chương trình ta thấy đạo hàm cấp 2 đổi dấu trên khoảng này.
Tương tự trên $(1;2)$ với $\varepsilon = 0.5 \times 10^{-5} = 0.000005$, hàm f cũng không thỏa mãn điều kiện áp dụng định lý 3
Tuy nhiên, để ý một chút, nếu ta thu nhỏ khoảng phân ly lại, chỉ xét f trên khoảng $(1,5;2)$ thay vì $(1;2)$ thì các điều kiện của định lý 3 lại thỏa mãn



Hình 26: Chương trình chạy ví dụ 12

Như vậy, ta tìm được một nghiệm là $x = 1.67963$ sau 3 lần lặp

Ví dụ 13

Tìm tất cả các nghiệm của phương trình $\cos x - x^2 - x = 0$ với 5 chữ số đáng tin

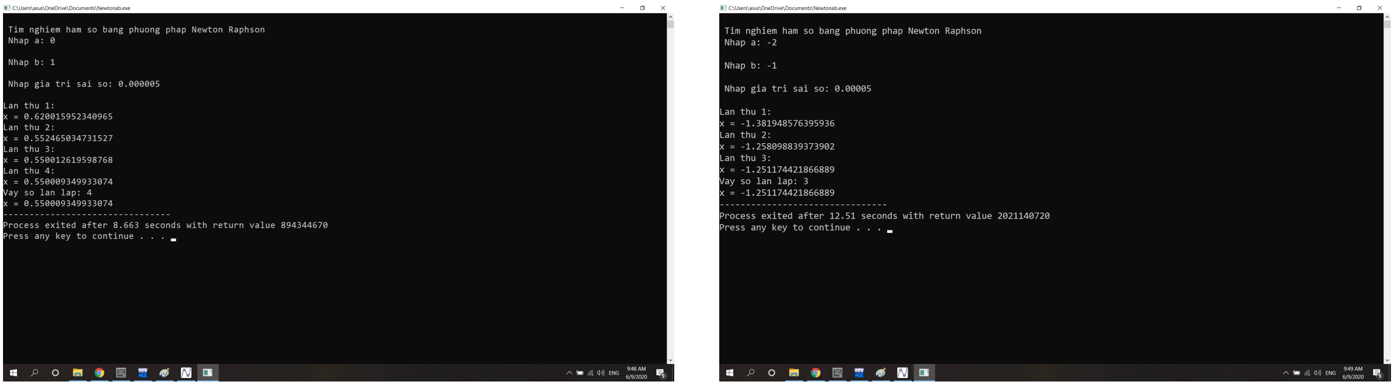
Đặt $f(x) = \cos x - x^2 - x$, tính toán

$$f(-2) = 2.4161;\; f(-1) = 0.5403;\; f(0) = 1;\; f(1) = -1.4597$$

và chứng minh rằng

$$f(x) < 0,\ \forall x < -2; \quad f(x) < 0,\ \forall x > 1$$

Suy ra $(-2;-1)$ và $(0;1)$ là hai khoảng phân ly nghiệm duy nhất của phương trình.
Trên đoạn $(0;1)$ chọn sai số $\varepsilon = 0.5 \times 10^{-5} = 0.000005$ ta tìm được $x = 0.55009$ sau 4 lần lặp
Trên đoạn $(-2;-1)$ với sai số $\varepsilon = 0.5 \times 10^{-4} = 0.00005$ được kết quả như trên
Vậy, phương trình có 2 nghiệm $x = 0.55009$ và $x = -1.2512$ với 5 chữ số đáng tin



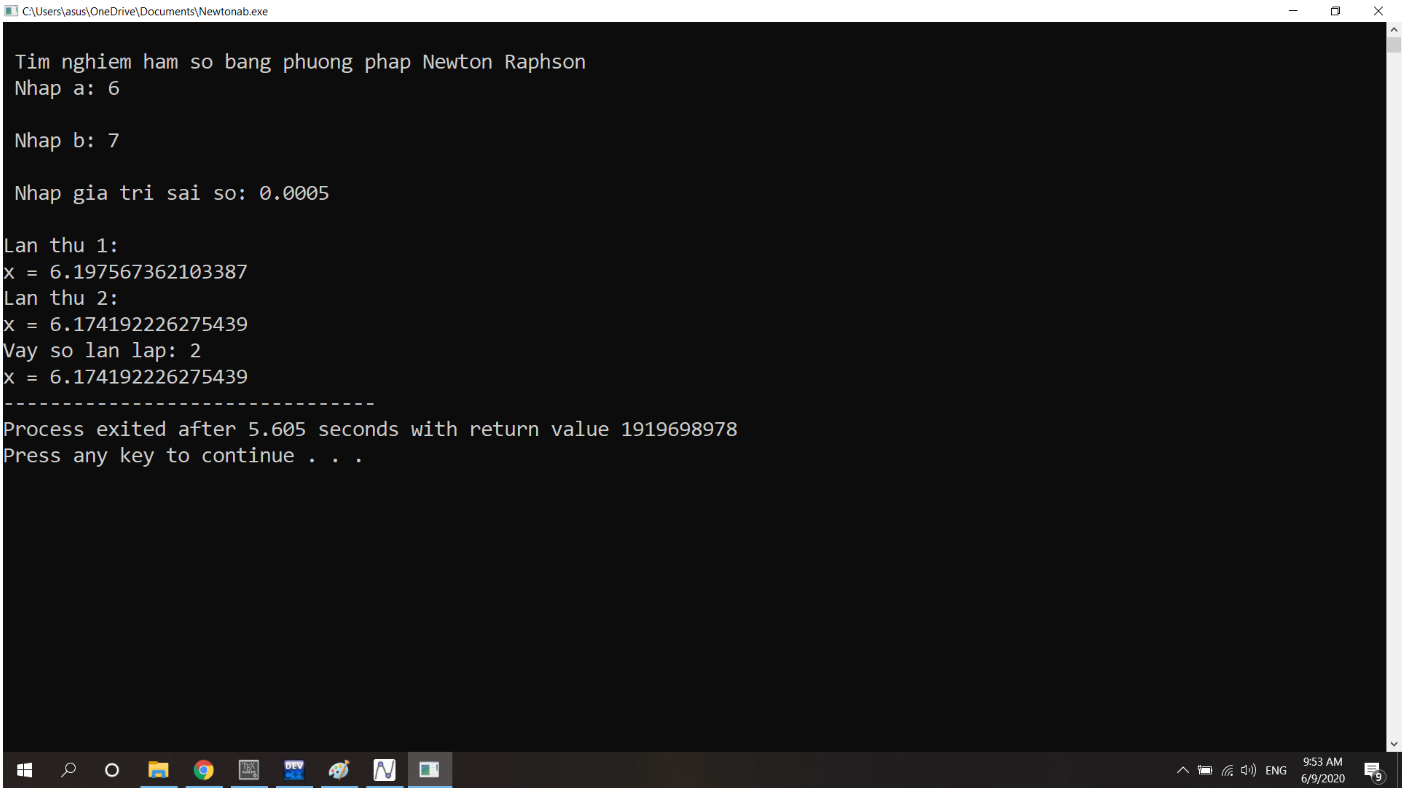
Hình 27: Chương trình chạy ví dụ 13

Ví dụ 14

Ta đã biết biểu thức $n!$ chỉ được xác định với $n \in \mathbb{N}$. Do đó với giá trị n đủ lớn, người ta thường xấp xỉ giai thừa bởi $f(n)$, ở đó

$$f(x) = (2\pi)^{\frac{1}{2}}\ x^{x+\frac{1}{2}}\ e^{-x}\left(1 + \frac{1}{12x} + \frac{1}{288x^2}\right)$$

Tìm x với 4 chữ số đáng tin sao cho $f(x) = 1000$



Hình 28: Chương trình chạy ví dụ 14

Ta viết lại phương trình

$$f(x) = (2\pi)^{\frac{1}{2}}\ x^{x+\frac{1}{2}}\ e^{-x}\left(1 + \frac{1}{12x} + \frac{1}{288x^2}\right) - 1000 = 0$$

Lấy logarith 2 vế ta có

$$g(x) = \frac{1}{2}\ln(2\pi) + \left(x + \frac{1}{2}\right)\ln x - x + \ln\left(1 + \frac{1}{12x} + \frac{1}{288x^2}\right) - 3\ln 10$$

Vì $6! = 720$ nên thử tính $g(6), g(7)$ ta thấy $(6; 7)$ là khoảng phân ly nghiệm của phương trình.
Để nghiệm có 4 chữ số đáng tin, ta cần có $\varepsilon = 0.5 \times 10^{-3} = 0.0005$.
Vậy phương trình có nghiệm $x = 6.1742$ sau 2 lần lặp

7.2 Các ví dụ thực tế

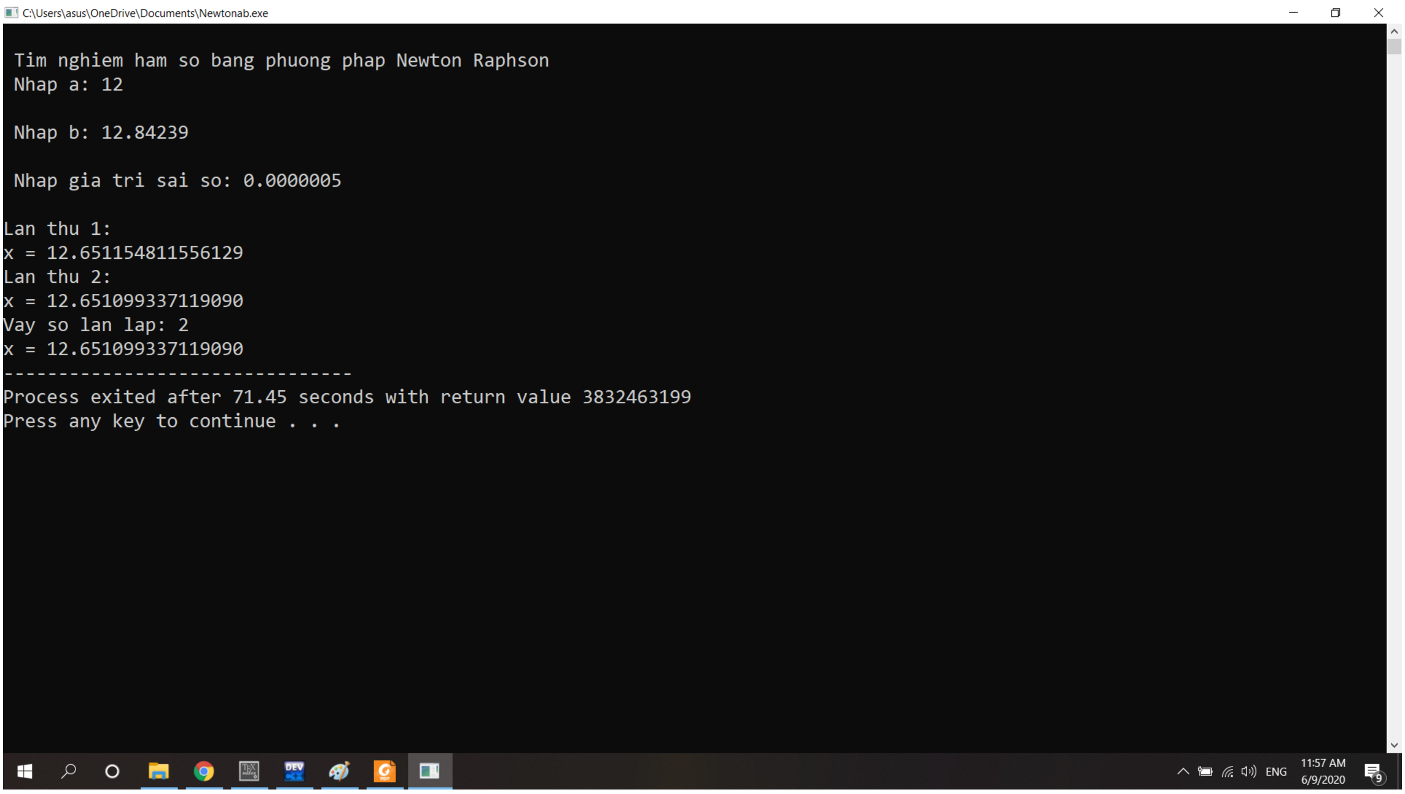
Ví dụ 15

Giả sử 1 mol khí Clo ở áp suất 2 atm và nhiệt độ 313 K. Cho biết với khí Clo, $R = 0.08206$ atm.l^2/mol.K, $a = 6.29$ atm.l^2/mol^2 và $b = 0.0562$ l/mol. Tìm thể tích của khí?

Phương trình *van der Waals* xác định mối liên hệ giữa các thông số trạng thái của khối khí thực

$$\left(P + \frac{n^2 a}{V^2}\right)(V - nb) = nRT$$

trong đó P là áp suất, V là thể tích khối khí, T là nhiệt độ tuyệt đối, n là số mol, R là hằng số các chất khí và a là hệ số tỷ lệ phụ thuộc bản chất của chất khí, b là số bổ chính về thể tích, là 2 hằng số thực nghiệm



Hình 29: Chương trình chạy ví dụ 15

Ta viết lại phương trình *van der Waals* thành hàm theo V

$$f(V) = \left(P + \frac{n^2 a}{V^2}\right)(V - nb) - nRT = \left(2 + \frac{2.69}{V^2}\right)(V - 0.0562) - 25.684784$$

Suy ra đạo hàm theo V

$$2 + \frac{6.29}{V^2} - \frac{12.58}{V^3}(V - 0.0562)$$

Xuất phát từ phương trình khí lý tưởng, ta đi tìm V_0

$$V_0 = \frac{nRT}{P} = \frac{1 \times 0.08206 \times 313}{1} = 12.84239 \text{ (l)}$$

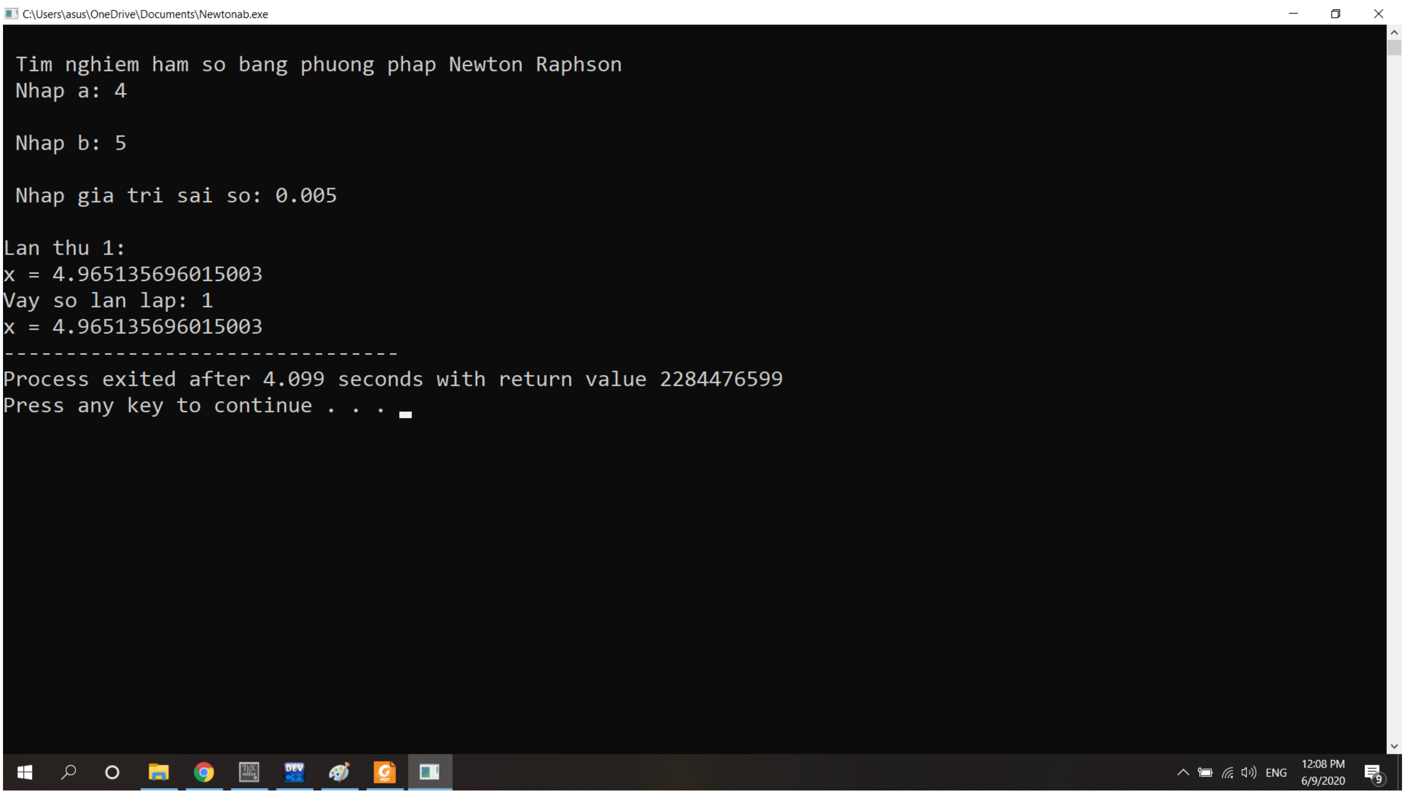
Từ đó, tính $f(12)$ và $f(13)$, ta tìm được khoảng phân ly $(12; 12.84239)$.
Tự chọn sai số $\varepsilon = 0.0000005$, ta thu được kết quả $V = 12.65109$, nhỏ hơn khoảng 1.5% thể tích ở trạng thái lý tưởng. Trong bài này ta chỉ cần 2 lần lặp!

Ví dụ 16

Biết rằng mật độ năng lượng ψ trong một vật đen trong trạng thái cân bằng nhiệt ở một nhiệt độ xác định tuân theo định luật bức xạ *Planck*

$$\psi = \frac{8\pi ch\lambda^{-5}}{e^{\frac{ch}{\lambda kT}} - 1}$$

trong đó λ là bước sóng bức xạ, T là nhiệt độ tuyệt đối của vật đen, h là hằng số *Planck*, k là hằng số *Boltzmann* và c là tốc độ ánh sáng.
Tìm bước sóng mà tại đó mật độ năng lượng là lớn nhất?



Hình 30: Chương trình chạy ví dụ 16

Để tìm bước sóng làm cực đại mật độ năng lượng, ta tính

$$\frac{d\psi}{d\lambda} = \frac{8\pi ch\lambda^{-6}}{e^{\frac{ch}{\lambda kT}} - 1}\left(-5 + \frac{\frac{ch}{\lambda kT}\, e^{\frac{ch}{\lambda kT}}}{e^{\frac{ch}{\lambda kT}} - 1}\right)$$

Bây giờ ta đi giải phương trình $\dfrac{d\psi}{d\lambda} = 0$.
Để ý rằng khi $\lambda \to 0$ và khi $\lambda \to \infty$ thì $\dfrac{8\pi ch\lambda^{-6}}{e^{\frac{ch}{\lambda kT}} - 1} \to 0$, nhưng tại λ này mật độ năng lượng là nhỏ nhất
Như vậy, mật độ năng lượng là lớn nhất khi nhân tử trong tích trên bằng 0, hay

$$1 - \frac{ch}{5\lambda_{\max} kT} = e^{\frac{ch}{\lambda kT}}$$

ở đó kí hiệu $\lambda_{\max}$ là bước sóng làm cực đại ψ.
Đặt $x = \dfrac{ch}{\lambda_{\max} kT}$, viết lại phương trình

$$1 - \frac{x}{5} = e^{-x}$$

Xét $f(x) = 1 - \dfrac{x}{5} - e^{-x}$ có $f'(x) = -e^{-x} + \dfrac{1}{5}$. Vì đường thẳng $1 - \dfrac{x}{5}$ cắt Ox tại $x = 5$ và $e^{-5} \simeq 6.74 \times 10^{-3}$ nên ta có thể đoán f có nghiệm gần $x = 5$.
Tính $f(4)$ và $f(6)$ ta tìm được khoảng phân ly nghiệm $(4; 5)$.
Tự chọn $\varepsilon = 0.005$, ta tìm được $x \simeq 4.965$, suy ra

$$\lambda_{\max} = \frac{ch}{4.965kT}$$

Ví dụ 17

Một người lao động muốn dành tiền trong N năm để có được một khoản tiền là A bằng cách gửi tiết kiệm bổ sung hàng năm. Mỗi năm anh ta gửi vào tài khoản một khoản tiền là P. Hỏi lãi suất ngân hàng tối thiểu là bao nhiêu thì anh ta mới thực hiện được kế hoạch này?

Giả sử lãi suất ngân hàng là x, thì $0 < x < 1$
Hết năm đầu, số tiền của anh trong ngân hàng là $P + xP = (1 + x)P$
Sau đó gửi thêm tiền nên trong ngân hàng có $P + (1 + x)P$
Hết năm 2, số tiền trong ngân hàng là $(1 + x)\left[P + (1 + x)P\right]$
Sau đó gửi thêm P nên tiền trong ngân hàng là $P + (1 + x)\left[P + (1 + x)P\right]$
Tương tự, hết năm 3 thì trong ngân hàng có

$$(1 + x)\left\{P + (1 + x)\left[P + (1 + x)P\right]\right\}$$

Đặt $y = 1 + x,\ (1 < y < 2)$, rút gọn ta được là

$$yP + y^2P + y^3P$$

Một cách tổng quát, giả sử sau N năm đủ số tiền A thì

$$P\left(y^N + y^{N-1} + \ldots + y^2 + y\right) \geq A$$

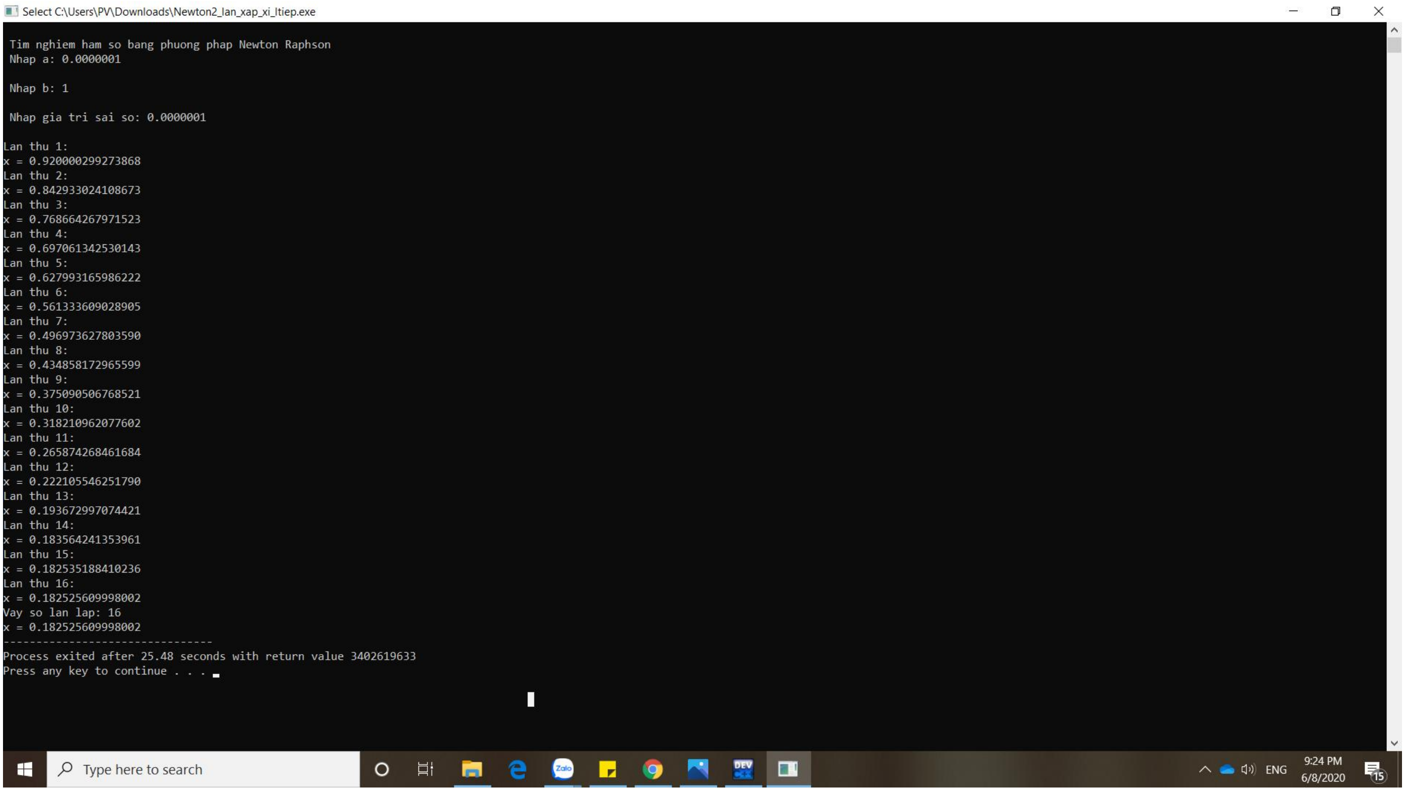
Suy ra

$$P\,\frac{(1+x)^{N+1}-x-1}{x} \geq A$$

Ta đi giải phương trình

$$f(x) = P\,\frac{(1+x)^{N+1}-x-1}{x} - A = 0$$

Vì x cần tìm thỏa mãn $0 < x < 1$ nên một cách tự nhiên, ta chọn luôn $(0;1)$ làm khoảng phân ly của f. Tuy nhiên, nếu làm như vậy, ta lại gặp vấn đề là f không xác định tại $x = 0$.
Để khắc phục điều này, ta chọn khoảng phân ly là $(10^{-7}, 1)$ thay vì $(0;1)$ mà không lo bị mất nghiệm do x là lãi suất nên nó luôn lớn hơn 10^{-7}
Chọn $A = 1.5$ tỷ, $P = 3$ triệu, $N = 25$ năm và sai số $\varepsilon = 10^{-7}$, ta được kết quả $x = 0.182525609$ sau 16 lần lặp



Hình 31: Chương trình chạy ví dụ 17

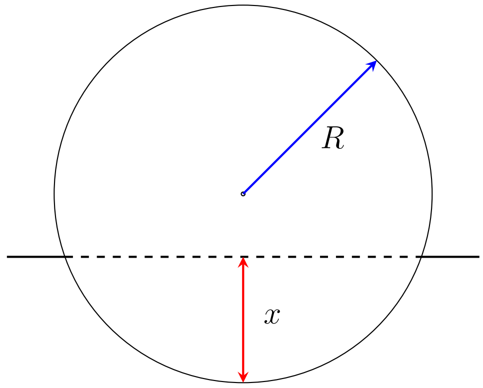
1. Nếu ta biến đổi f về dạng phương trình

$$g(x) = P(1+x)^{N+1} - Px - P - Ax = 0$$

thì lại không dùng được phương pháp $Newton$ do đạo hàm g' đổi dấu trên khoảng $(0;1)$
2. Hàm g trên còn có bất lợi là không triệt tiêu được A trong biểu thức g', làm cho việc khảo sát khó khăn hơn do A là đại lượng chưa biết, nó có thể lớn tùy ý.
3. Vì x là lãi suất tối thiểu nên nghiệm r tìm được phải thỏa mãn $f(r) > 0$. Nếu $f(r) < 0$ thì ta có thể tìm vô số r' rất gần r để xấp xỉ nghiệm của f mà vẫn thỏa mãn ε. Rõ ràng r' là lựa chọn tối ưu hơn r. Nếu A lên cỡ vài nghìn tỷ, thì sai lệch r và r' có thể lên đến hàng chục triệu!
Nếu tìm được $f(r) < 0$, ta có thể dùng dây cung vì 2 phương pháp này sinh ra trong cùng điều kiện, và 2 dãy lặp tiến về nghiệm theo hướng ngược nhau

Ví dụ 18

Một quả bóng bán kính $R = 5.5$ cm và trọng lượng riêng là 0.6 được thả trôi trên mặt nước. Tính chiều cao phần quả bóng bị chìm dưới nước?



Hình 32: Quả bóng thả trôi trên mặt nước

Theo định luật 3 của *Newton*, mọi lực tác dụng đều có một phản lực ngược chiều cùng độ lớn. Trong trường hợp này, quả bóng sẽ cân bằng khi trọng lực cân bằng với lực đẩy *Archimedes*

$$P = F_A$$

Trọng lượng của quả bóng được tính theo công thức

$$P = m\,g = V\,\rho_b\,g = \left(\frac{4}{3}\pi R^3\right)\rho_b\,g$$

trong đó, $V = \frac{4}{3}\pi R^3$ là thể tích quả bóng, ρ_b là mật độ khối lượng của quả bóng và g là gia tốc

Trọng lượng của quả bóng phải bằng với trọng lượng phần nước bị chiếm chỗ, được tính theo công thức

$$\pi x^2\left(R - \frac{x}{3}\right)\rho_w\,g$$

trong đó, $\pi x^2\left(R - \frac{x}{3}\right)$ là thể tích phần quả bóng bị chìm trong nước, ρ_w là mật độ khối lượng của nước và g là gia tốc

Thay vào ta có

$$\left(\frac{4}{3}\pi R^3\right)\rho_b\,g = \pi x^2\left(R - \frac{x}{3}\right)\rho_w\,g \Rightarrow 4R^3\frac{\rho_b}{\rho_w} - 3x^2R + x^3 = 0$$

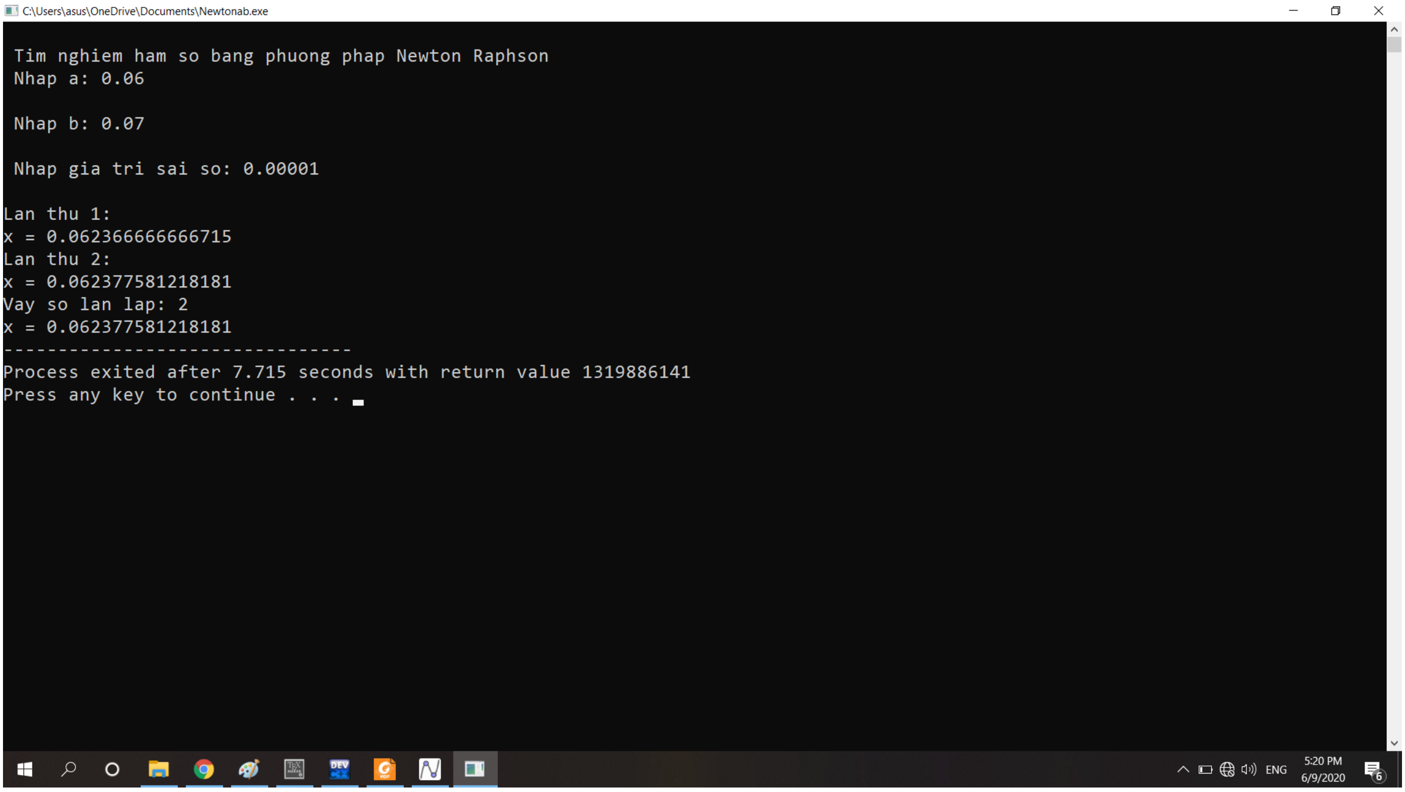
ở đó, trọng lượng riêng của quả bóng γ_b được tính bởi $\gamma_b = \frac{\rho_b}{\rho_w}$

Thay $R = 5.5$ cm $= 0.055$ m, $\gamma_b = 0.6$ suy ra

$$x^3 - 0.165x^2 + 3.993 \times 10^{-4} = 0$$

Từ thực tiễn, do x là chiều cao phần quả bóng chìm dưới nước nên ta có $x \in (0; 0.11)$.
Tính toán trực tiếp, ta tìm được một khoảng phân ly $(0.06; 0.07)$.

Chọn sai số $\varepsilon = 0.00001$, thu được kết quả như dưới đây

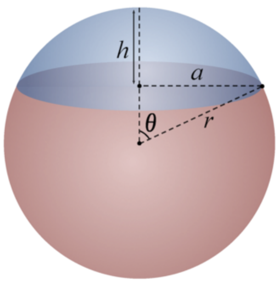


Hình 33: Chương trình chạy ví dụ 18

Như vậy, ta tìm được $x = 0.0623775$ sau 2 lần lặp.

Chú ý: Trong bài toán này ta đã sử dụng công thức tính thể tích chỏm cầu

$$V = \frac{\pi h^2}{3}(3r - h)$$



Hình 34: Chỏm cầu màu đỏ và màu xanh

Công thức này được chứng minh dễ dàng bằng tích phân.

7.3 Tìm nghịch đảo của một số

Ví dụ 19

Tính $\frac{1}{3}$ bằng phương pháp *Newton* với 3 chữ số đáng tin sau dấu phẩy

Ta viết lại thành bài toán giải phương trình $3 - \dfrac{1}{x} = 0$ với $\varepsilon = 0.5 \times 10^{-3} = 0.0005$
Trong phần lý thuyết, ta đã chỉ ra rằng có thể chọn bất cứ điểm x_0 nào trong khoảng $\left(0, \dfrac{2}{3}\right)$ làm xấp xỉ đầu để đảm bảo thuật toán hội tụ. Hơn thế nữa, nếu $a < 1$ thì ta có thể chọn ngay $x_0 = 2^{-m}$, với m thỏa mãn $a = 2^m x_1$.
Trong bài toán này ta có $a = 3 > 1$ nhưng vẫn tìm x_0 có dạng $x_0 = 2^{-m}$, với $3 = 2^m x_1$ để hy vọng dãy lặp hội tụ nhanh hơn. Áp dụng công thức lặp sau

$$x_{n+1} = x_n(2 - 3x_n)$$

1. Ở bài toán này, người dùng chỉ cần nhập vào số muốn tìm nghịch đảo và sai số đầu vào.
2. Ta không cần kiểm tra các điều kiện khoảng phân ly nghiệm (a, b) và đạo hàm f', f'' không đổi dấu trên khoảng (a, b).
Vì đã khảo sát trực tiếp điều kiện hội tụ cho bài toán này, nên công việc lập trình giảm đi rất nhiều, bỏ qua được bài toán tìm giá trị lớn nhất, nhỏ nhất!

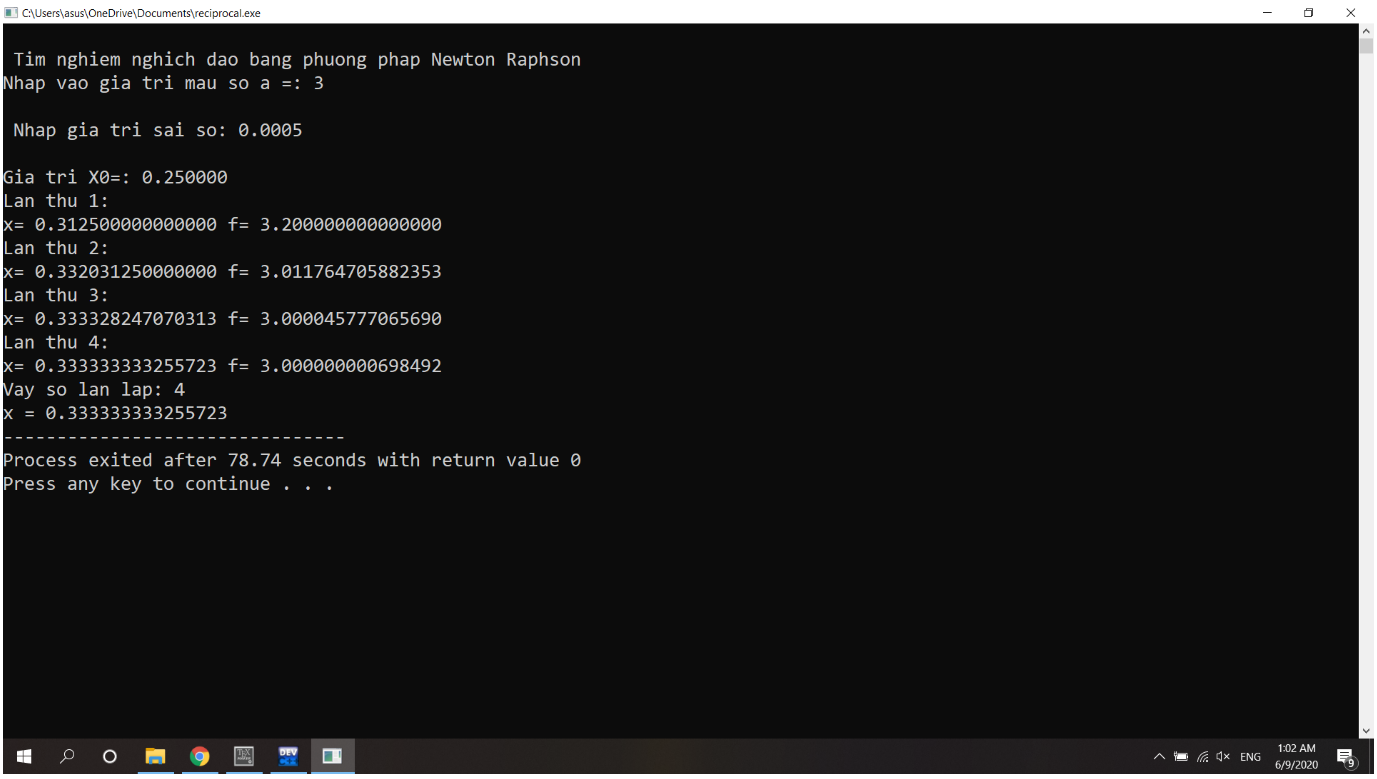
Ta sử dụng điều kiện dừng là $|x_n - x_{n-1}| < \varepsilon$. Chương trình giải:

```c
#include<stdio.h>
#include<math.h>

double f(double x)
{
    return 1/x;
}

int sign(double a)
{
    if (a<0)
    {
        return -1;
    }
    else
    {
        return 1;
    }
}

int main()
{
    printf("\n Tim nghiem nghich dao bang phuong phap Newton Raphson");
    int m=0;
    double Xo, Xn, a, a0, e;
    printf("\nNhap vao gia tri mau so a =: ");
    scanf("%lf", &a);
    printf("\n Nhap gia tri sai so: ");
    scanf("%lf", &e);
    int signa = sign(a);
    a = sign(a) * a; //Nhan a voi dau cua chinh no do chi can xet a >= 0
    a0 = a;
    if (a0 > 1 || a0 < 0.5)
    {
        while (a0 > 1) //Neu a0 > 1 thi chia cho 2 de duoc dang a = 2^m a1, sao cho
    0.5 < a1 < 1
        {
```

```
            a0 = a0 / 2;
            m += 1;
        }
        while (a0 < 0.5) //Neu a0 < 0.5 thi nhan voi 2 de duoc dang a = 2^m a1, sao cho 0.5 < a1 < 1
        {
            a0 = a0 * 2;
            m -= 1;
        }
    }
    Xo = pow(2,-m); //x0 = 2^-m
    printf("\nGia tri X0=: %lf", Xo);
    int i=0;
    do
    {
        i += 1;
        Xn = Xo * (2 - a * Xo); //Phep lap x_{n+1} = x_n(2 - a x_n)
        printf("\nLan thu %d:\n", i);
        printf("x= %8.15lf", signa*Xn);
        printf("f= %8.15lf", f(signa*Xn));
        if(fabs(Xn-Xo) < e)
        {
            printf("\nVay so lan lap: %d \n", i);
            printf("x = %8.15lf", signa*Xn);
            return 0;
        }
        else Xo = Xn;
    } while (fabs(Xn-Xo) < e);
}
```

Listing 7: Chương trình giải bài toán tìm nghịch đảo của một số dương

Màn hình kết quả



Hình 35: Kết quả chương trình tìm nghịch đảo của một số bằng phương pháp Newton

Bảng kết quả

$x_0 = 0.25$		
Lần lặp	x_i	$f(x_i)$
1	0.312500	3.200000
2	0.332031	3.011764
3	0.333328	3.000045
4	0.333333	3.000000

7.4 Mở rộng: Trường hợp nghiệm bội

Bài toán. Phương pháp Newton trong trường hợp nghiệm bội.

1. Nghiệm bội là gì? Ở phần lý thuyết, ta đã bàn về phương pháp tiếp tuyến để để tìm nghiệm đơn r của phương trình $f(x) = 0$, nghĩa là $f(r) = 0$, $f'(r) \neq 0$. Xét trường hợp

$$f'(r) = f''(r) = \ldots = f^{(m-1)}(r) = 0 \neq f^{(m)}(r)$$

Áp dụng khai triển *Taylor* suy ra

$$f(x) = (x-r)^m . \frac{f^m(\xi_x)}{m!}$$

ở đó ξ_x nằm giữa r và x. Nếu ta đặt $g(x) = \dfrac{f^m(\xi_x)}{m!}$, thì

$$f(x) = (x-r)^m q(x)$$

với q là hàm liên tục tại r và $g'(r) \neq 0$.
Do đó, khi x trong lân cận của r, hàm $f(x)$ "hành xử" giống như một đa thức có nghiệm bội m tại r. Bởi lý do này, ta nói rằng r là nghiệm bội m của f.

2. Trong trường hợp này, về lý thuyết, tất cả các định lý về phương pháp *Newton* đều không thể sử dụng được, vì chúng đều có điều kiện là r là nghiệm đơn của f, hay $f'(r) \neq 0$.

Tuy nhiên, trong nhiều bài toán, ngay cả khi $f'(r) = 0$, ta vẫn có thể dùng công thức 2!

Ví dụ: Giải phương trình $(x-1)^3 = 0$

Gọi $f(x) = (x-1)^3$, ta có $f'(x) = 3(x-1)^2$. Dễ thấy f nhận $r = 1$ làm nghiệm bội 3. Công thức của dãy lặp:

$$\begin{aligned} x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)}, \qquad n \geq 0 \\ &= x_n - \frac{(x_n-1)^3}{3(x_n-1)^2} \\ &= x_n - \frac{x_n - 1}{3} \\ &= \frac{2x_n + 1}{3} \end{aligned}$$

Trong ví dụ này, mặc dù $f'(1) = 0$ nhưng ta hoàn toàn tránh được việc "chia cho không", công thức dãy $\{x_n\}$ hoàn toàn xác định. Nếu chọn x_0 hợp lý, phương pháp *Newton* vẫn sẽ hội tụ.

3. Trong trường hợp nghiệm bội, nếu sử dụng phương pháp tiếp tuyến thì sẽ có những khó khăn như sau:

Hội tụ chậm hơn nhiều so với trường hợp nghiệm đơn

Cụ thể, ta sẽ chứng minh phương pháp $Newton$ hội tụ tuyến tính trong tình huống này.
Có $f'(x) = m(x-r)^{(m-1)}q(x) + (x-r)^m q'(x)$ suy ra

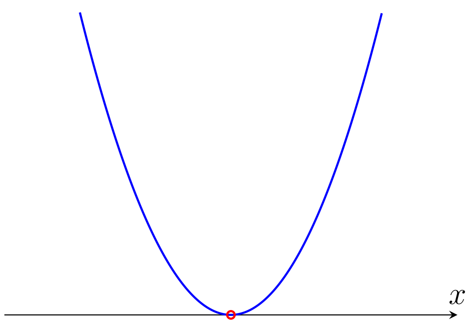
$$g(x) = x - \frac{(x-r)^m q(x)}{m(x-r)^{(m-1)}q(x) + (x-r)^m q'(x)} = x - \frac{(x-r)q(x)}{mg(x) - (x-r)q'(x)}$$

Suy ra g xác định tại $x = r$ và $g(r) = r$, hay r là điểm bất động của g. Mặt khác, tính trực tiếp ta có

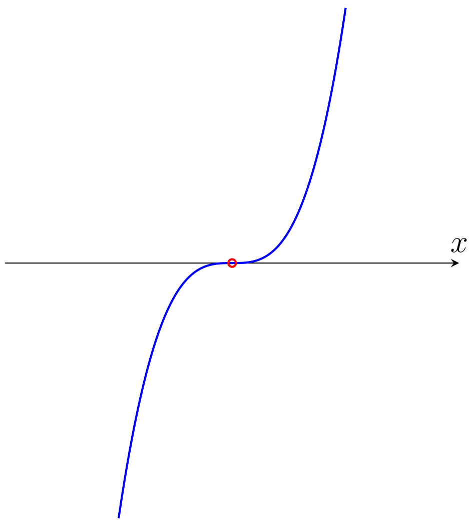
$$g'(r) = 1 - \frac{1}{m}$$

Điều này có nghĩa là thuật toán $Newton$ sẽ hội tụ nếu chọn x_0 đủ gần r. Khi đó, nó hội tụ với tốc độ tuyến tính với $g'(r) = 1 - \dfrac{1}{m}$.

1. Nếu $m = 2$, hay r là nghiệm kép, thì $g'(r) = \dfrac{1}{2}$, phương pháp $Newton$ có tốc độ hội tụ tương đương với phương pháp chia đôi.
2. Trong trường hợp $m > 2$, phương pháp $Newton$ chậm hơn phương pháp chia đôi.
3. Ta cũng có thể hình dung được điều này từ hình vẽ trên đây. Ta thấy rằng cả 2 đường cong đều khá "phẳng" trong lân cận của nghiệm r, và như đã phân tích trong phần lý thuyết đầu tiên, điều này làm cho sự hội tụ chậm lại.



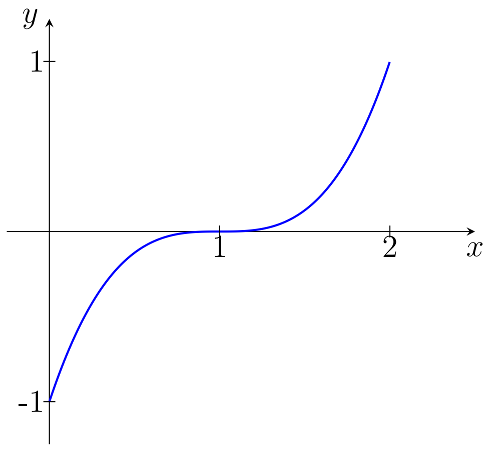
Hình 36: $f(x) = (x-1)^2$



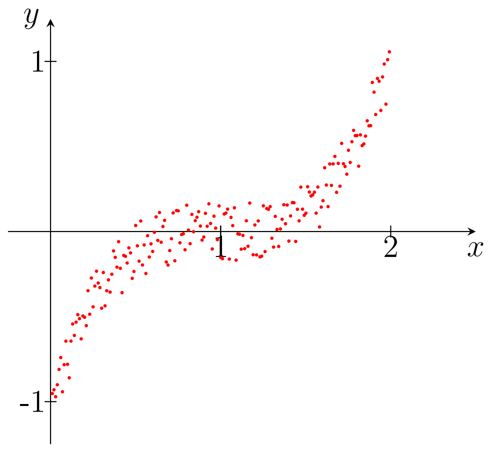
Hình 37: $f(x) = (x-1)^3$

Khoảng nhiễu không chắc chắn khá lớn

Một trong những hệ quả tiêu cực mà ta có thể thấy ngay của việc làm tròn số khi tính giá trị của hàm $f(x)$ bằng máy tính là ta sẽ thu được một hàm xấp xỉ $\hat{f}(x)$ không liên tục, khi quan sát đồ thị $\hat{f}(x)$ ở một độ chia đủ nhỏ.
Mỗi phép toán dùng khi tính $f(x)$ đều sẽ sinh ra sai số. Nếu xét đến sự sai lệch của các sai số này, ta thu được một giá trị $\hat{f}(x)$, với sai sô $f(x) - \hat{f}(x)$ là một số số nhỏ ngẫu nhiên khi x thay đổi. Sai số này được gọi là nhiễu.



Hình 38: Đường cong hàm $f(x) = (x-1)^3$



Hình 39: Sai số nhiễu hàm $f(x) = (x-1)^3$

Khi quan sát đồ thị của $\hat{f}(x)$ ở một độ chia đủ nhỏ, ta thấy nó là một tập các điểm rời rạc lộn xộn. Điều này ảnh hưởng đến các tính toán khác sử dụng hàm $f(x)$. Ví dụ, tìm nghiệm của $f(x)$ thông qua $\hat{f}(x)$ sẽ dẫn đến một khoảng nhiễu không chắc chắn trong lân cận nghiệm. Với những trường hợp nghiệm bội, khoảng nhiễu không chắc chắn này càng lớn.

4. Ưu thế của phương pháp *Newton* khi giải bài toán nghiệm bội.

Giả sử r là nghiệm bội m của phương trình $f(x) = 0$. Ta xét trường hợp m là số chẵn để thấy rõ ưu điểm của phương pháp *Newton*

- Phương pháp chia đôi không sử dụng được do ta không thể tìm được khoảng phân ly nghiệm r. Nếu chọn khoảng (a, b) chỉ chứa nghiệm r, thì f không đổi dấu trên khoảng này. Nếu chọn được khoảng (a, b) có $f(a)f(b) < 0$, thì f có nghiệm trên (a, b), phương pháp chia đôi hội tụ. Tuy nhiên nó sẽ hội tụ về nghiệm khác r.

- Không có cơ sở đảm bảo phương pháp dây cung hội tụ. Do hàm f không đổi dấu trong lân cận (a, b) của r, nên dây cung đi qua các điểm $\big(a, f(a)\big)$, $\big(b, f(b)\big)$ cắt trục hoành tại giao điểm nằm ngoài (a, b). Ta không thể suy ra dãy $\{x_n\}$ hội tụ, nhưng cũng không có cơ sở để kết luận nó phân kỳ.

- Phương pháp lặp đơn hội tụ nếu ta chọn được hàm $g(x)$ thỏa mãn các điều kiện chặt chẽ. Tuy nhiên ta đều biết rằng việc kiểm tra các điều kiện này tương đối khó và phải tùy vào từng trường hợp cụ thể.

- Nếu chọn xấp xỉ đầu x_0 hợp lý, phương pháp *Newton* hội tụ về nghiệm r. Sử dụng công thức gốc 2, tốc độ hội tụ có thể chậm hơn tốc độ của phương pháp chia đôi, tuy nhiên, sau đây, ta chỉ ra rằng có thể khôi phục được tốc độ hội tụ bậc hai.

5. Khôi phục tốc độ hội tụ bậc hai trong trường hợp nghiệm bội. Để làm được điều này, ta dựa trên cơ sở lý thuyết của phương pháp lặp đơn.

Thay đổi hàm đầu vào rồi dùng công thức *Newton*

Giả sử $f(x)$ có r là nghiệm bội m, nghĩa là $f(x) = (x-r)^m q(x)$, với $\lim\limits_{x\to r} q(x) \neq 0$.

Xét hàm $F(x) = \dfrac{f(x)}{f'(x)}$, ta có

$$F(x) = \frac{(x-r)q(x)}{(x-r)q'(x) + mq(x)} = (x-r)h(x),$$

ở đó $\lim_{x\to r} h(x) = \dfrac{1}{m} \neq 0$. Do đó, r là nghiệm đơn của $F(x)$, nên F sẽ hội tụ về r với tốc độ bậc hai. Thay $F(x) = \dfrac{f(x)}{f'(x)}$ vào công thức hàm lặp của phương pháp *Newton* ta có

$$g(x) = x - \frac{F(x)}{F'(x)} = x - \frac{f(x)/f'(x)}{\dfrac{\left[f'(x)\right]^2 - f(x)f''(x)}{\left[f'(x)\right]^2}}$$
$$= x - \frac{f(x)f'(x)}{\left[f'(x)\right]^2 - f(x)f''(x)}$$

Như vậy ta có công thức

$$x_{n+1} = x_n - \frac{f(x_n)f'(x_n)}{\left[f'(x_n)\right]^2 - f(x_n)f''(x_n)}$$

1. Phương pháp này có nhược điểm là ở mỗi lần lặp, tốn 3 lần gọi hàm để tính giá trị f, f', f''.
2. Phương pháp này thường được dùng khi ta không biết giá trị của m

Ví dụ 20

Xét phương trình

$$1 + \ln x - x = 0$$

có $x = 1$ là ngiệm bội 2. Chọn $x_0 = 2$ và $\varepsilon = 10^{-5}$.

Sử dụng công thức lặp trên với điều kiện dừng là sai số tương đối đủ nhỏ $\left|\dfrac{x_n - x_{n-1}}{x_n}\right| < \varepsilon$

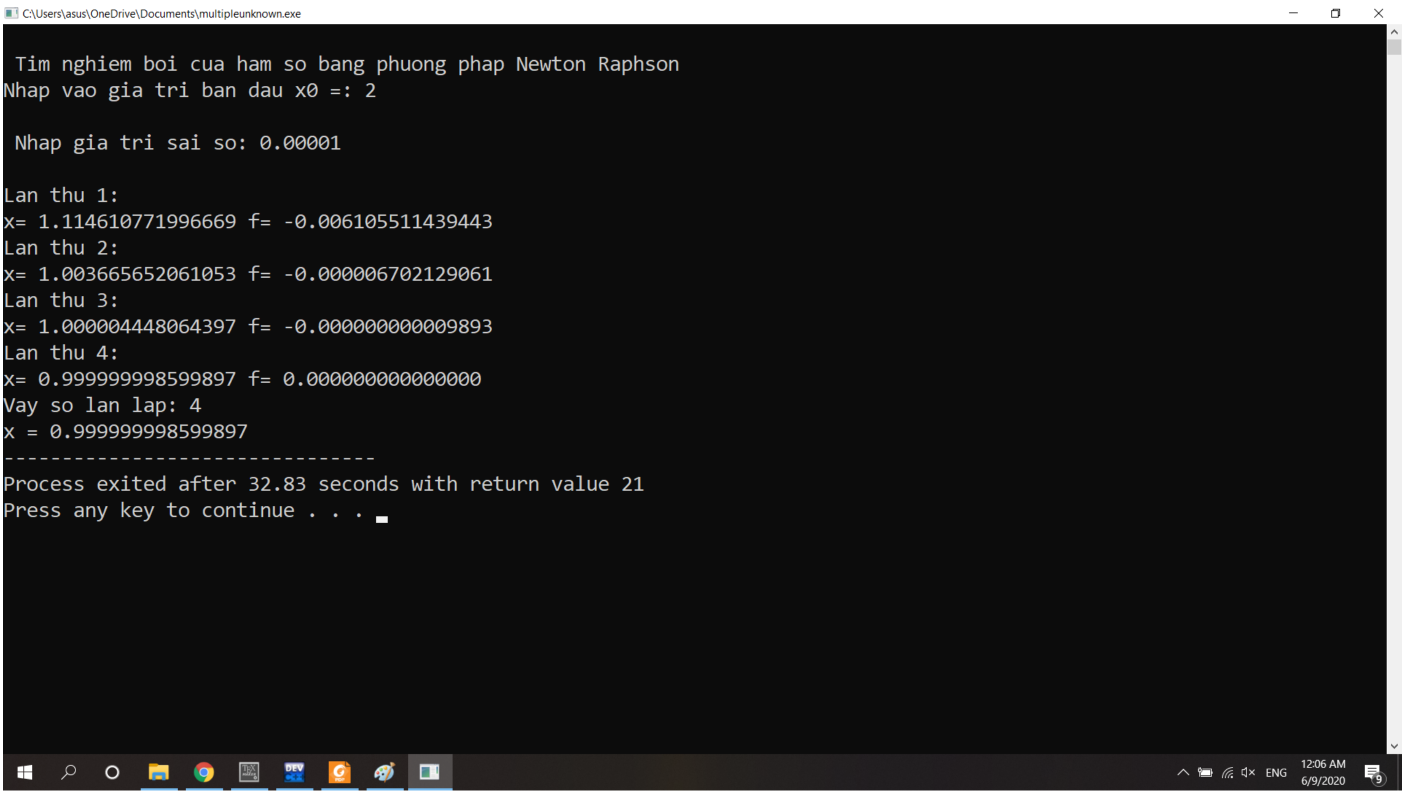
Chương trình:

```
#include<conio.h>
#include<stdio.h>
#include<math.h>

double f(double x)
{
  return 1 + log(x) - x;
}
double df(double x)
{
  double h = 1e-7;
  return ((f(x + h) - f (x - h))/(2 * h));
}
double ddf(double x)
{
  float h = 1e-3;
  return ((df(x + h) - df(x - h))/(2 * h));
}

int main()
{
  printf("\n Tim nghiem boi cua ham so bang phuong phap Newton Raphson");
  int i = 1;
  double x0, s, e;
  printf("\nNhap vao gia tri ban dau x0 =: ");
```

```c
  scanf("%lf", &x0);
  printf("\n Nhap gia tri sai so: ");
  scanf("%lf", &e);
  do
  {
    s = x0;
    x0 = x0 - f(x0) * df(x0)/(pow(df(x0),2) - ddf(x0) * f(x0));
    printf("\nLan thu %d:\n", i);
    printf("x= %8.15lf", x0);
    printf(" f= %8.15lf", f(x0));
    i++;
  }while(fabs((s - x0)/x0) > e);

  if(fabs((s - x0)/x0) < e){
    printf("\nVay so lan lap: %d \n", i-1);
    printf("x = %8.15lf", x0);
  }
}
```

Listing 8: Trường hợp nghiệm bội không biết giá trị m

Màn hình kết quả



Hình 40: Giải phương trình $1 + \ln x - x = 0$ không biết số bội

Bảng kết quả

Lần lặp	x_i	$f(x_i)$
1	1.114610	-0.006105
2	1.003665	-0.000006
3	1.000004	-9.893×10^{-12}
4	0.999999	0.000000

Thay đổi hàm lặp $g(x)$ của phương pháp *Newton*

Ta biết rằng phương pháp *Newton* gốc chỉ hội tụ tuyến tính tới nghiệm bội $m > 1$ vì $g'(r) = 1 - \dfrac{1}{m}$ khác 0. Số $\dfrac{1}{m}$ xuất hiện khi tính giá trị đạo hàm của đại lượng $\dfrac{f(x)}{f'(x)}$ trong công thức lặp. Điều này gợi ý cho ta nhân thêm m vào $\dfrac{f(x)}{f'(x)}$ và cải tiến công thức 2 thành công thức

$$\widetilde{g}(x) = x - m.\frac{f(x)}{f'(x)}$$

Ta dễ dàng kiểm tra $\widetilde{g}'(r) = 1 - m.\dfrac{1}{m} = 0$, suy ra dãy lặp hội tụ với tốc độ bậc hai. Như vậy, ta đã chứng minh được

Định lý 12

Dãy lặp sinh bởi công thức

$$x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)} \tag{13}$$

hội tụ với tốc độ bậc hai.

1. Phương pháp này có ưu điểm là ở mỗi lần lặp, không cần tính thêm hàm gì mới so với công thức gốc.
2. Phương pháp này thường được dùng khi ta biết trước giá trị của m.
3. Trong một số trường hợp, m được xấp xỉ bởi số nguyên gần nhất với

$$m \approx \frac{1}{1 - \dfrac{x_n - x_{n-1}}{x_{n-1} - x_{n-2}}}$$

Ví dụ 20

Xét phương trình

$$1 + \ln x - x = 0$$

có $x = 1$ là ngiệm bội 2. Chọn $x_0 = 2$ và $\varepsilon = 10^{-5}$.

Sử dụng công thức lặp trên với điều kiện dừng là sai số tương đối đủ nhỏ $\left|\dfrac{x_n - x_{n-1}}{x_n}\right| < \varepsilon$

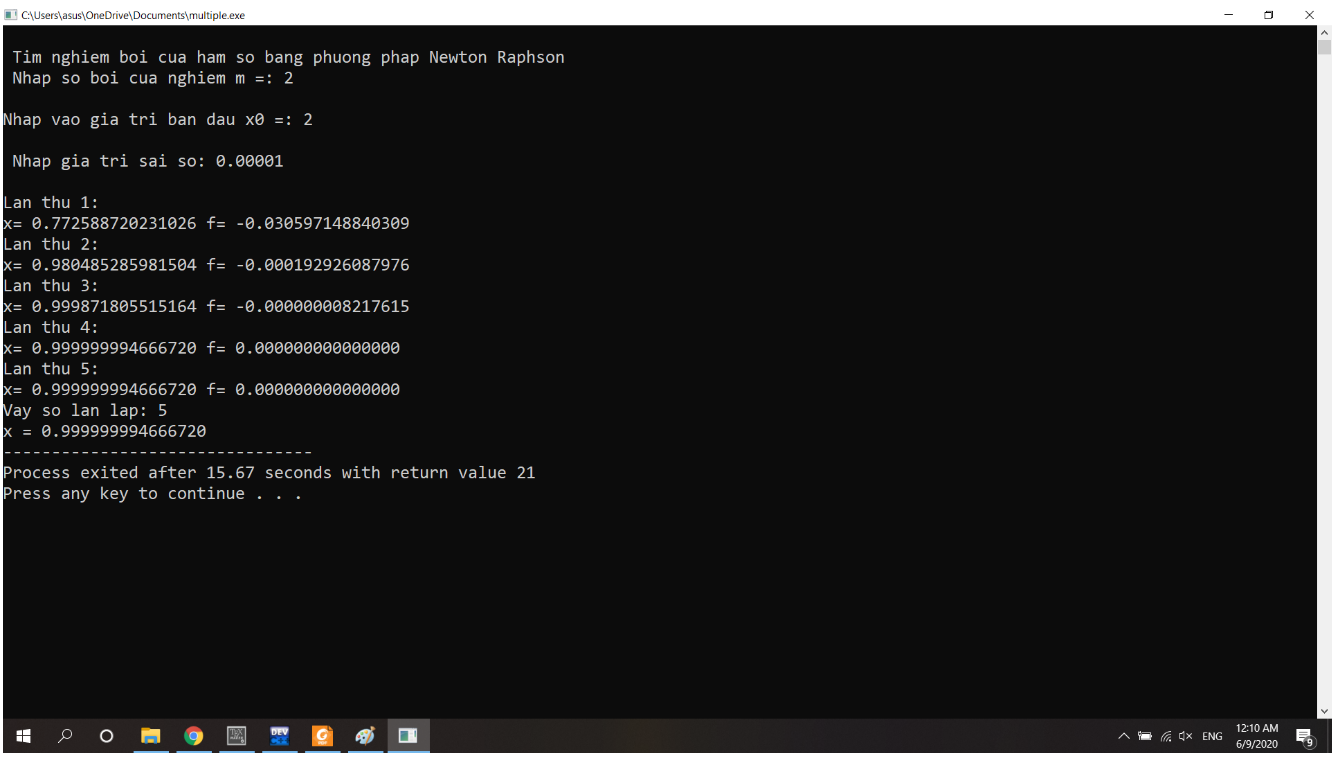
Chương trình:

```
#include<conio.h>
#include<stdio.h>
#include<math.h>

double f(double x)
{
  return 1 + log(x) - x;
}
double df(double x)
{
  double h = 1e-7;
```

```c
  return ((f(x + h) - f (x - h))/(2 * h));
}

int main()
{
  printf("\n Tim nghiem boi cua ham so bang phuong pháp Newton Raphson");
  int m, i = 1;
  double x0, s, e;
  printf("\n Nhap so boi cua nghiem m =: ");
  scanf("%d", &m);
  printf("\nNhap vao gia tri ban dau x0 =: ");
  scanf("%lf", &x0);
  printf("\n Nhap gia tri sai so: ");
  scanf("%lf", &e);

  do
  {
    s = x0;
    x0 = x0 - m*f(x0)/df(x0);
    printf("\nLan thu %d:\n", i);
    printf("x= %8.15lf", x0);
    printf(" f= %8.15lf", f(x0));
    i++;
  }while(fabs((s - x0)/x0) > e);

  if(fabs((s - x0)/x0) < e){
    printf("\nVay so lan lap: %d \n", i-1);
    printf("x = %8.15lf", x0);
  }
}
```

Listing 9: Trường hợp nghiệm bội biết giá trị m

Màn hình kết quả



Hình 41: Giải phương trình $1 + \ln x - x = 0$ khi biết số bội

Bảng kết quả

Lần lặp	x_i	$f(x_i)$
1	0.772588	-0.030597
2	0.980485	-0.000192
3	0.999871	-8.2176×10^{-9}
4	0.999999	0.000000
5	0.999999	0.000000

8 Mở rộng: Phương pháp lai

Ý tưởng của phương pháp

1. Một số hạn chế của phương pháp *Newton*

   - Xuất phát từ công thức 2, nếu chọn x_0 không hợp lý ta có thể rơi vào các trường hợp phân kì, vòng lặp vô hạn hoặc "chia cho không"
   - Trong trường hợp nghiệm bội, phương pháp *Newton* theo công thức 2 cũng gặp nhiều khó khăn
   - Sử dụng định lý 3 để lập trình, ta thấy rằng việc kiểm tra điều kiện đầu vào của thuật toán *Newton* khá phức tạp và đòi hỏi nhiều công việc

   Do đó, trên thực tế, trong nhiều trường hợp, người ta thường sử dụng các phương án kết hợp.

2. Các phương pháp lai hay được sử dụng

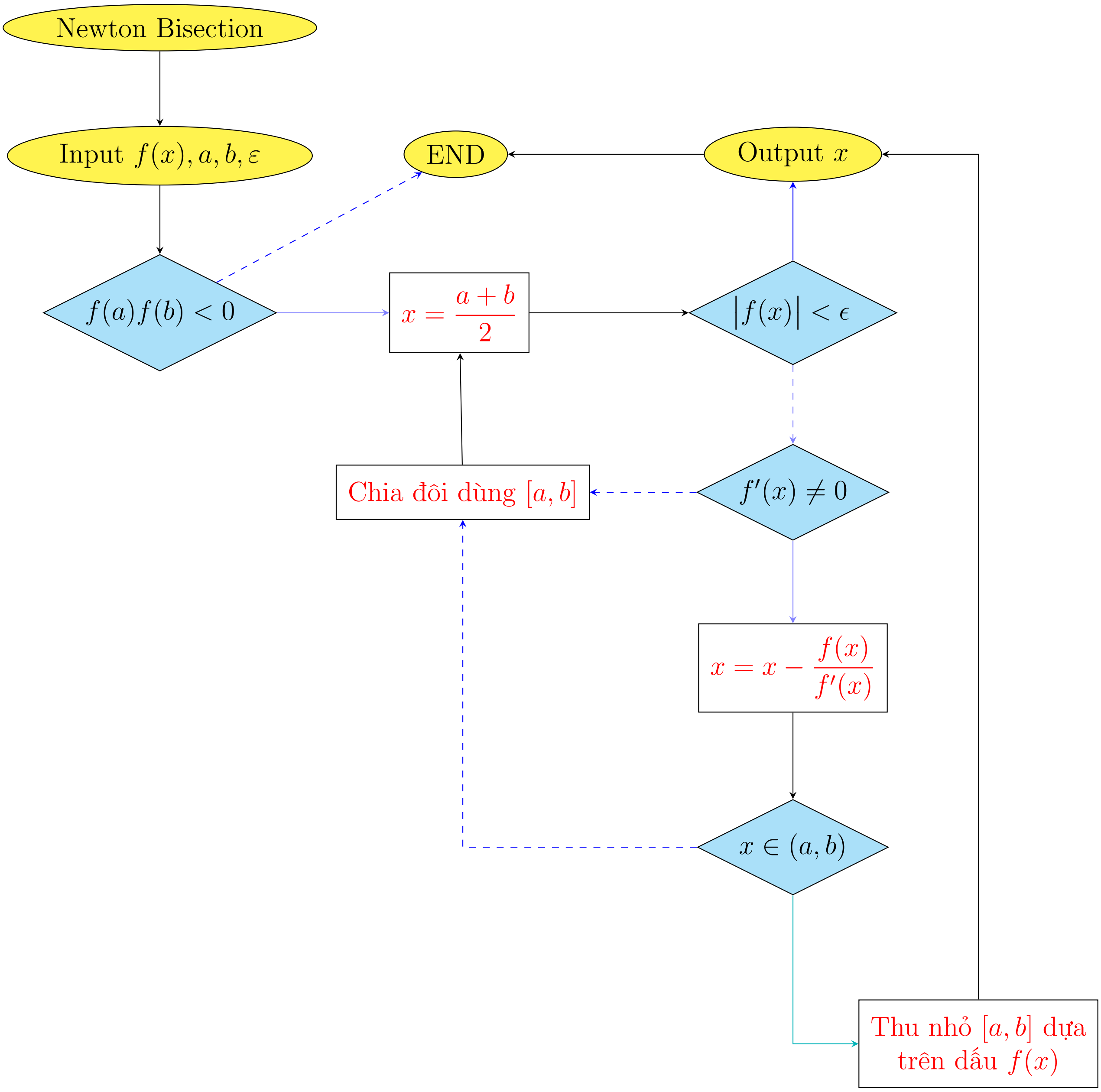
   - Tính đạo hàm cấp một f': Phương pháp *Newt - Safe*
   - Tính đạo hàm cấp hai f'': Phương pháp *Halley*, còn gọi là *Hal - Safe*
   - Tính các đạo hàm cấp cao: Phương pháp *Householder*
   - Trong trường hợp f là đa thức, ta có phương pháp *Horner*, còn gọi là *Horn - Safe*
   - Nếu không sử dụng đạo hàm, ta có các phương pháp *Steffesen*, phương pháp *Dekker*, phương pháp *Brent*

3. Trong mục này ta xem xét phương pháp kết hợp *Newton* với *Bisection*, ta tạm gọi là phương án lai *Newt - Safe*.

4. Ý tưởng của thuật toán này là nếu giao điểm của tiếp tuyến với trục hoành nằm ngoài đoạn $[a, b]$ hoặc tại x_n là điểm cực trị của hàm, hay khi phương pháp tiếp tuyến rút ngắn khoảng cách từ điểm lặp đến nghiệm chưa đủ nhanh, ta sẽ sử dụng phương pháp chia đôi, còn nếu giao điểm nằm bên trong $[a, b]$ và $f' \neq 0$ và phương pháp *Newton* đủ tốt, ta tiếp tục sử dụng công thức của dãy lặp *Newton*.

5. Phương pháp này là sự kết hợp giữa yếu tố "chắc chắn" của phương pháp chia đôi với tốc độ hội tụ nhanh của phương pháp tiếp tuyến.

Thuật toán



Hình 42: Sơ đồ khối thuật toán kết hợp Newton với chia đôi (*Newt - Safe*)

Ví dụ 21

Giải phương trình $\ln x - 1 = 0$ với 5 chữ số đáng tin sau dấu phẩy bằng phương pháp *Newt - Safe*

Ta sử dụng điều kiện dừng là $|f(x_n)| < \varepsilon$ với $\varepsilon = 0.000005$. Chương trình giải:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

double f(double x)
{
    return log(x) - 1 ;
}

```

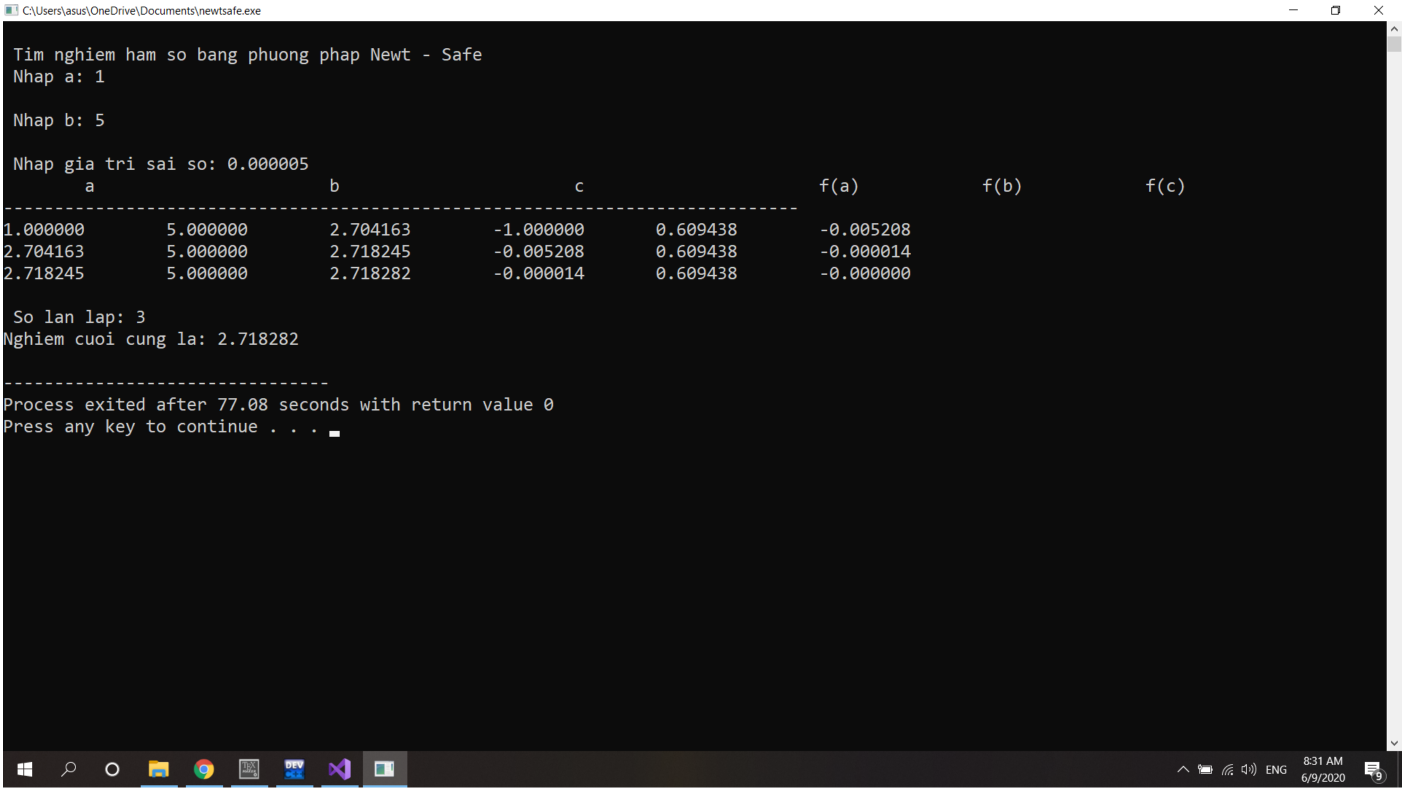
```
double df(double x)
{
  double h = 1e-7;
  return ((f(x + h) - f(x - h))/(2 * h));
}

int sign(double a)
{
    if (a<0)
    {
        return -1;
    }
    else
    {
        return 1;
    }
}

int main(int argc, const char * argv[])
{
  printf("\n Tim nghiem ham so bang phuong phap Newt - Safe");
    double a, b, eps, x, oldx;
    char hbar[90] = {[0 ... 77] = '-'};
    int i = 0;
    do{
    printf("\n Nhap a: ");
    scanf("%lf", &a);
    printf("\n Nhap b: ");
    scanf("%lf", &b);

    if (sign(f(a)) == sign(f(b))) {
      printf("Day khong phai khoang cach ly nghiem! Hay nhap lai!");
    }
    if (a == b) {
      printf("a phai khac b! Hay nhap lai!");
    }
  } while ((sign(f(a)) == sign(f(b))) || (a == b));

  if (a > b)
    {
        a=a+b;
        b=a-b;
        a=a-b;
    }

    printf("\n Nhap gia tri sai so: ");
    scanf("%lf", &eps);
    printf("\ta\t\t\tb\t\t\tc\t\t\tf(a)\t\tf(b)\t\tf(c)\n");
    printf("%s\n", hbar);

    oldx = b;
    x = (a + b)/2;

    do
    {
        if (fabs(f(x)) < eps)
        {
            printf ("Nghiem cua phuong trinh la: %lf",x);
            break;
        }
```

```
        if (df(x) == 0) //Neu gap diem cuc tri
        {
            printf("\nGap phai diem cuc tri nen dung phuong phap chia doi");
            x = (a + b)/2;
            printf("%lf\t%lf\t%lf\t", a, b, x);
            printf("%lf\t%lf\t%lf\n", f(a), f(b), f(x));
            if (f(a) * f(x) < 0)
            {
                oldx = b;
                b = x;
            }
            else
            {
                oldx = a;
                a = x;
            }
        }

        oldx = x;
        if (a < (x - f(x)/df(x)) && (x - f(x)/df(x)) < b) //Neu giao diem cua tiep
    tuyen nam trong (a,b)
        {
            x = x - f(x)/df(x);
            printf("%lf\t%lf\t%lf\t", a, b, x);
            printf("%lf\t%lf\t%lf\n", f(a), f(b), f(x));
            if (f(a) * f(x) < 0)
            {
                b = x;
            }
            else
            {
                a = x;
            }
        }
        else //Neu giao diem cua tiep tuyen nam ngoai (a,b)
        {
            printf("\nCat o ngoai (a,b) nen dung phuong phap chia doi");
            x = (a + b)/2;
            printf("%lf\t%lf\t%lf\t", a, b, x);
            printf("%lf\t%lf\t%lf\n", f(a), f(b), f(x));
            if (f(a) * f(x) < 0)
            {
                b = x;
            }
            else
            {
                a = x;
            }
        }
        i++;

        if (fabs(f(x)) < eps) //Dung khi f(x_n) du nho
        {
            break;
        }
    } while(1);
    printf("\n So lan lap: %d",i);
    printf("\nNghiem cuoi cung la: %lf\n",x);
    return 0;
```

```
}
```

Listing 10: Chương trình thuật toán kết hợp *Newton* với chia đôi

Màn hình kết quả



Hình 43: Giải phương trình $\ln x - 1 = 0$ bằng phương pháp lai

Bảng kết quả

a	b	c	$f(a)$	$f(b)$	$f(c)$
1	5	2.704163	-1	0.609438	-0.005208
2.704163	5	2.718245	-0.005208	0.609438	-0.000014
2.718245	5	2.718282	-0.000014	0.609438	-0.000000

Như vậy, sau 3 lần lặp ta nhận được nghiệm $x = 2.718282$

9 Các chương trình sử dụng thêm

Chương trình giải các ví dụ phần lý thuyết

```
#include<conio.h>
#include<stdio.h>
#include<math.h>

//sin x;  x_0 = 7.539822;  n = 5;
//sin(x); x0 = 7.539822; n = 5;

//xe^{-x};  x_0 = 2;  n = 15;
//x * exp(-x); x0 = 2; n = 15;

//arctan x;  x_0 = 1.45;  n = 10;
//atan(x); x0 = 1.45; n = 10;

double f(double x)
{
  return sin(x);
}
double df(double x)
{
  double h = 1e-7;
  return ((f(x + h) - f (x - h))/(2 * h));
}

int main()
{
  printf("\n Tim nghiem ham so bang phuong phap Newton Raphson");
  int i;
  int n = 5;
  double x0, x1;
  printf("\nNhap vao gia tri ban dau x0 =: ");
  scanf("%lf", &x0);

  for(i = 1; i <= n; i++)
  {
    x1 = x0 - f(x0)/df(x0);
    x0 = x1;
    printf("\nLan thu %d:\n", i);
    printf("x= %8.15lf", x1);
    printf(" f= %8.15lf", f(x1));
  }
}
```

Listing 11: Chương trình thuật toán *Newton* (minh họa 3 ví dụ lý thuyết)

Chương trình phương pháp chia đôi

```
#include <stdio.h>
#include <math.h>

double f(double);

int sign(double);

```

```c
double f(double x){
    return log(x) - 1;
}

int sign(double x){
    if(x >= 0)
        return 1;
    return -1;
}

int main(){
    double a, b, c, eps;
    int i = 0;

    printf("\n Tim nghiem ham so bang phuong phap chia doi");

    do{
    printf("\n Nhap a: ");
    scanf("%lf", &a);
    printf("\n Nhap b: ");
    scanf("%lf", &b);

    if (sign(f(a)) == sign(f(b))) {
      printf("Day khong phai khoang cach ly nghiem! Hay nhap lai!");
    }
    if (a == b) {
      printf("a phai khac b! Hay nhap lai!");
    }
  } while ((sign(f(a)) == sign(f(b))) || (a == b));

  if (a > b)
    {
        a=a+b;
        b=a-b;
        a=a-b;
    }

    printf("\n Nhap gia tri sai so: ");
    scanf("%lf", &eps);

  char hbar[90] = {[0 ... 88] = '-'};
    printf("\ta\t\t\tb\t\t\tc\t\t\tf(a)\t\tf(b)\t\tf(c)\n");
    printf("%s\n", hbar);

    while(fabs(b - a) > eps){ //Dieu kien dung |b_n - a_n| ≤ ε
        c = (a + b) / 2.0;

        printf("%lf\t%lf\t%lf\t", a, b, c);
        printf("%d\t%d\t%d\n", sign(f(a)), sign(f(b)), sign(f(c)));
        if(sign(f(c)) == sign(f(a)))
            a = c;
        else
            b = c;
        i++;
    }
    printf("%d iterations\n\n", i);
}
```

Listing 12: Chương trình thuật toán chia đôi giải ví dụ 7

Chương trình phương pháp tìm giá trị lớn nhất, nhỏ nhất

```
#include<conio.h>
#include<stdio.h>
#include<math.h>

double f(double x)
{
  return log(x) - 1;
}

double df(double x)
{
  double h = 1e-7;
  return ((f(x + h) - f(x - h))/(2 * h));
}

//Gia tri nho nhat cua mot ham
double min(double f(double x), double a, double b)
{
  double alpha, x0 = a, e = 1e-7;
  double mini = a;

  alpha = (b - a)/10000;

  do
  {
    int loop = 10000;
    x0 = x0 + e;
    if(df(x0) > 0)
      do
      {
            x0 = x0 + alpha*df(x0);
            loop--;
            if(loop < 0) break;
      }while(fabs(df(x0)) > e);
    else
    {
      do
      {
            x0 = x0 - alpha*df(x0);
            loop--;
            if(loop < 0) break;
      }while(fabs(df(x0)) > e);
    }
    if (x0 > b) break;
    else
    {
      if (f(x0) < f(mini))
      mini = x0;
      }
    }while(1);

    if (f(mini) < f(b)) return f(mini);
    else return f(b);
}

//GTLN cua mot ham
double nef(double x)
{
```

```
  return -1 * f(x);
}
double max(double f(double x), double a, double b)
{
  return -1 * min(nef,a,b);
}

int main()
{
  printf("\n Tim GTLN GTNN cua ham so bang phuong phap Steepest Descent");
  double a, b;

  printf("\n Nhap a: ");
  scanf("%lf", &a);
  printf("\n Nhap b: ");
  scanf("%lf", &b);

  double m, n;
  m = max(f,a,b);
  n = min(f,a,b);

  printf("\nmax = %8.15lf\n", m);
  printf("min = %8.15lf\n", n);
}
```

Listing 13: Chương trình thuật toán tìm min max dùng Steepest Descent

10 Tổng kết

Phương pháp *Newton* là một trong những phương pháp rất quan trọng trong việc giải gần đúng nghiệm của phương trình $f(x) = 0$. Nó có những ưu điểm nổi bật hơn so với việc sử dụng các phương pháp chia đôi, dây cung hay lặp đơn. Mặc dù độ phức tạp trong tính toán của phương pháp *Newton* cao hơn các phương pháp trên do phải tính đạo hàm tại mỗi bước, nhưng với tốc độ hội tụ bậc hai, các chữ số chính xác tăng nhanh khoảng 2 lần sau mỗi lần lặp. Do đó, nhìn chung số lần lặp sẽ giảm đi, đổi lấy cái giá của việc khối lượng tính toán lớn ở mỗi bước lặp là hoàn toàn xứng đáng.

Tuy nhiên không phải lúc nào chúng ta cũng có thể sử dụng phương pháp này. Khi sử dụng phương pháp *Newton* chúng ta cần hết sức lưu ý đến việc kiểm tra những điều kiện đầu vào của hàm và chọn đúng điểm *Fourier*. Mặt khác, khi tất cả các điều kiện đầu vào đều thỏa mãn, chúng ta cũng cần cân nhắc, bởi lẽ không phải với tất cả mọi bài toán, phương pháp *Newton* cũng tốt hơn phương pháp chia đôi, dây cung hay lặp đơn. Ta cần có cái nhìn rõ ràng và thực tế nhất đối với từng bài toán và từng phương pháp để có được lời giải tối ưu nhất.

Phương pháp *Newton* cũng được ứng dụng từ sớm và đóng góp một phần quan trọng trong các thế hệ máy tính, giúp máy tính thực hiện phép chia mà không cần định nghĩa phép chia bằng cách xấp xỉ nghiệm của phương trình $a - \dfrac{1}{x} = 0$ với $a > 0$, hay nói cách khác đây chính là bài toán tìm nghịch đảo của một số.

Tài liệu tham khảo

1. Bài giảng Giải tích số, TS Hà Thị Ngọc Yến, Viện Toán ứng dụng và Tin học, Đại học Bách Khoa Hà Nội, 2018
2. Giải tích số, Lê Trọng Vinh, Nxb. Khoa học kĩ thuật, 2007
3. A Friendly Introduction to Numerical Analysis, Brian Bradie, Person Publisher, 2006
4. Computational Mathematics, B. P. Demidovich, I. A. Maron, MIR Publisher, 1981
5. Numerical Algorithms, Justin Solomon, CRC Press Publisher, 2015
6. Numerical Analysis, Richard Burden, J. Douglas Faires, Brooks/Cole Publisher, 2011
7. Numerical Analysis, Timothy Sauer, Person Publisher, 2012
8. Afternotes on Numerical Analysis, G. W. Stewart, SIAM, 1996
9. Numerical Methods (Problems and Solutions), M. K. Jain, S.R.K. Iyengar, R.K. Jain, New Age International Publisher, 2009
10. Numerical Methods , S.R.K. Iyengar, R.K. Jain, New Age International Publisher, 2009
11. An Introduction To Numerical Analysis, Kendal E. Atkinson, John Wiley & Sons Publisher, 1978
12. Applied Numerical Analysis, Gerald Wheatley, Person Publisher, 2004
13. Numerical Analysis for Scientists and Engneers: Theory and C Programs, Madhumangal Pal, Vidyasagar University
14. Numerical Recipes in C, William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, Cambridge University Press, 1992
15. Numerical Methods in Engineering and Science, B. S. Grewal, New Delhi, 2019
16. Numerical Mathematical Analysis, James B. Scarborough, Oxford & IBH Publisher, 1980
17. Numerical Mathematics and Computing, Ward Chenry, David Kingcaid, Thompson Brooks/Cole Publisher, 2008
18. Lectures of Numerical Analysis, T. Gambill, University of Illinois at Urbana - Champaign, 2012
19. Lecture notes for Introduction to Numerical Computation, Wen Shen, Penn State University
20. Numerical Methods with Applications, Autar K Kaw, Egwu Eric Kalu, 2008
21. Lectures of Numerical Methods, Thomas Bingham, Oregon Insitute of Technology
22. Lectures of Numerical Analysis, Chad Higdon-Topaz, William University
23. https://www.gnu.org/software/gsl/
24. https://nptel.ac.in/course.html