

Successive approximations: Divide and Average, Interval Bisection, and Newton's Method

Searching for solutions to various equations has been a central activity for mathematicians from earliest times. Several outcomes of this search are **Completing the square** and the **quadratic formula** as methods for calculating the exact solution to a quadratic equation are two results of this search. **Cardan's Formula** for solution to cubic equations is another. However, solutions to really messy equations may have no algebraic method of solution. For example, the equation $2^x - \sin x = x^3$ cannot be solved algebraically. It can be solved numerically (using the solve or nsolve commands) or graphically (using zero or intersection).

So just how did mathematicians solve this type of equation before the widespread availability of our present technology? One class of techniques early mathematicians developed was that of successive approximations. These are repetitive techniques that involve repeating a process over and over, until the desired degree of precision is attained. Their repetitive nature makes them excellent candidates for implementation on a computer. We will investigate three of these so-called **iterative techniques**.

- ❑ Divide and average
- ❑ Interval Bisection,
- ❑ Newton's Method

Divide and Average

It is believed that the first use of this method was by the Ancient Babylonians. They used it for finding irrational square roots, like $\sqrt{2}$. Here is how it went:

<u>Step</u>	<u>possible work:</u>
1. Estimate a value The estimate should be something you are confident is reasonably close to the answer. It may be found by guessing: $1^2 = 1$ and $2^2 = 4$ so $\sqrt{2}$ is between 1 and 2, probably closer to 1. Try 1.2.	Step 1. 1.2
2. Divide 2 (from $\sqrt{2}$) by this guess to get the next estimate	Step 2. $\frac{2}{1.2} \approx 1.67$
3. Average the previous est. and the current estimate	Step 3. $\frac{1.2 + 1.67}{2} \approx 1.435$
4. Repeat steps 2 and 3 until the current est. and the previous est. agree to the desired degree of accuracy	Step 2. $\frac{2}{1.435} \approx 1.3937$ Step 3. $\frac{1.3937 + 1.435}{2} \approx 1.41435$

(This is already correct to 3 decimal places)
(Repeat 2. & 3. as many times as needed)

This algorithm (or process) lends itself very nicely to the following program:

SqRoot(a)	When you execute this program, type the command, you type the
Prgm	radicand inside the parentheses. Ex: sqrt(2). The 2 is
Disp "Enter your guess:"	assigned to a. Otherwise, you would need an INPUT
Input g	statement for a in the program.
g/1.→g	This "trick" (notice the decimal point) of dividing by 1.
0→C	Start C at 0. It will "count" the number of iterations
Lbl L	forces g to be in decimal form, so that all future
a/g→r	outputs that depend on g will also be in decimal form.
(g+r)/2→n	
Disp n	Display the next approximation
C+1→C	Increase the count, C, by 1
Pause	
If abs(n-g)>1E-4 then	If current and previous estimates are .0001 or more apart,
n←g	use n as the guess, g, in the next iteration, and
GoTo L	go back and "divide and average again".
EndIf	
Disp "done"	
Disp "Iterations:",C	
EndProg	

Interval Bisection

Interval Bisection is an improvement on the Divide and Average technique. It can be used not only on square roots, but to find a real zero of any function. As we will see, it relies on the user's knowledge of the existence of a root between two known values of x , just like the BOUNDS we enter when using Max, Min, Zero, etc. in the Graph – Math menu.

The general technique is to

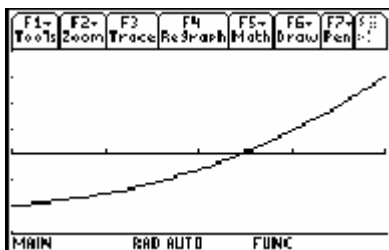
1. Locate a zero preferably between two consecutive integer values of x whose respective y -values have opposite signs. It is helpful to use the table feature to do this. The Intermediate Value Theorem guarantees that when you find two x -values whose respective y -values have opposite signs, and the function is continuous on the interval containing those two x -values, there is a zero between them, so these x -values are the two integers we need. Call them **lower** and **upper**. We will call the interval [**lower**, **upper**] the “search interval”
2. Find the midpoint of the interval
3. Decide which half of the interval (the right half or left half) the solution lies in by checking the sign of the function at the midpoint and comparing it to the signs of $f(a)$ and $f(b)$.
4. Make that half the new search interval
5. Return to step 2, and repeat steps 2 - 4 over and over until successive midpoints differ by less than the desired accuracy.

Example: Solve $x^3 - 2x = 1$.

Step

- 1a. Change to $f(x)$ form
- 1b. In the Y= editor, enter $f(x)$ for $y1$.
- 1c. Use a table to find two consecutive x values whose corresponding y -values have opposite signs. Call them “*lower*” and “*upper*”
2. Average *lower* and *upper*
3. Evaluate $f(x)$ at this value
4. Replace whichever of **lower** and **upper** whose functional value has the same sign as this result with this new x -value.
5. Repeat steps 2 – 4 as many times as necessary to get the desired accuracy.

Graph of $f(x) = x^3 - 2x - 1$
in the window $[1, 2] \times [-3, 4]$, $x\text{-scl} = \frac{1}{4}$



Process

$$f(x) = x^3 - 2x - 1$$

X	0	1	2	3
Y	-1	-2	3	20

So lower = 1, upper = 2
 $(1+2)/2=1.5$

$$f(1.5) = 1.5^3 - 2(1.5) - 1 = -.625$$

lower = 1, so $f(\mathbf{lower}) = f(1)$.

Now $f(1)$ and $f(1.5)$ have the same sign, so **lower** becomes 1.5.

Step 2: $(1.5 + 2)/2 = 1.75$

Step 3: $f(1.75) = .859375$

Step 4: *upper* changes to 1.75. because $f(1.75)$ and $f(\mathbf{upper})$ have the same sign

Step 2: $(1.5 + 1.75)/2 = 1.625$

Step 3: $f(1.625) = .041016$

Step 4: *upper* changes to 1.625 because $f(1.625)$ and $f(\mathbf{upper})$ have the same sign.

Step 2: $(1.5 + 1.625)/2 = 1.5625$

Step 3: $f(1.5625) = -0.3103$

Step 4: *lower* changes to 1.5625 because $f(1.5625)$ and $f(1.5)$ have the same sign

Step 2: $(1.5625 + 1.625)/2 = 1.59375$

Step 3: $f(1.59375) = -0.1393$

Step 4: *lower* changes to 1.59375 because $f(1.59375)$ and $f(1.5625)$ have the same sign

Etc. (After 12 more iterations (17 total), the result is 1.61803, correct to the nearest ten-thousandth.)

Here is a program – *Bisect()* – that will do this. It assumes that the equation is in proper form, and already entered in *y1*. It also assumes that you, the user, have already found appropriate values **L**(ower) and **u**(pper), with only one zero between **L** and **u**.

This program may produce erratic results if bounds are not chosen correctly.

```

bisect()
Prgm
Prompt L
Prompt u
Lbl a
    (L+u)/(2.)→m
    Disp m
    Pause
    If abs(y1(m))<1 E -005
        Goto e
    If y1(m)*y1(L)>0 Then
        m→L
        Goto a
    EndIf
    If y1(m)*y1(u)>0 Then
        m→u
        Goto a
    EndIf
Lbl e
Disp m
Disp "done"
EndPrgm

```

This “IF” line checks for the signs of $y1(m)$ and $y1(L)$. If their product is positive, they have the same sign, so the zero is between m and u . So the new L is the midpoint value, m .

This “IF” line checks for the signs of $y1(m)$ and $y1(u)$. If their product is positive, they have the same sign, and the zero is between L and m . So the new u is the midpoint value, m .

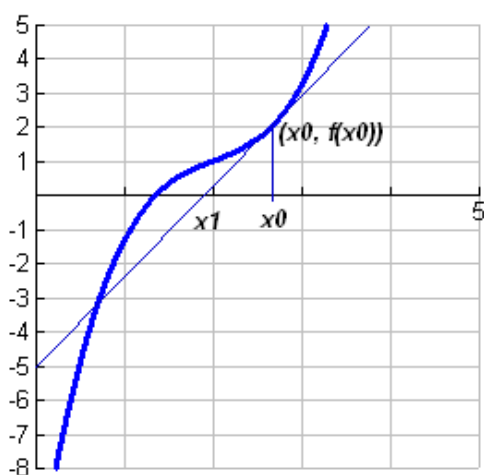
Newton's Method is another iterative method that frequently gives the desired degree of accuracy in fewer iterations than interval bisection. It makes use of *local linearity*. That is, **the idea that curves are locally straight**, and that consequently the tangent to a curve differs very little from the curve itself in the immediate neighborhood of the point of tangency

It works like this:

1. Pick an x -value that is “reasonably close” to the zero of the function. You should judge this from the appearance of the graph.
2. Using the derivative and point-slope form, write the equation of the tangent to the graph of the function at the x -value chosen in step 1.
3. Find the x -intercept of the tangent line whose equation you found in step 2.
4. Use this as the new x -value, and return to step 2, repeating steps 2 – 4 until the desired accuracy is achieved.

Here is the graph of $f(x) = (x - 2)^3 - 2 \sin(x - 2) + 3x - 5$ and its tangent line at the point $(x_0, f(x_0))$

First graphic iteration:



The slope, m , of the tangent to the graph at $(x_0, f(x_0))$ is $f'(x_0)$.

Using the point-slope form $y - y_1 = m(x - x_1)$ with slope m and the point $(x_0, f(x_0))$: we obtain:

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

Since we want the zero, which is the x -intercept, we let $y = 0$. Substituting 0 for y and solving for x :

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

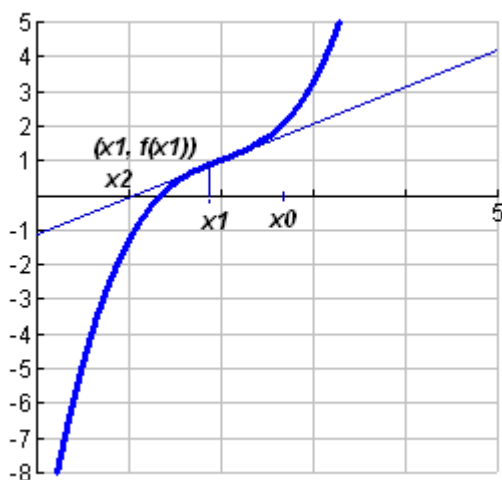
$$\frac{-f(x_0)}{f'(x_0)} = x - x_0$$

$$x = x_0 - \frac{f(x_0)}{f'(x_0)} \quad (*)$$

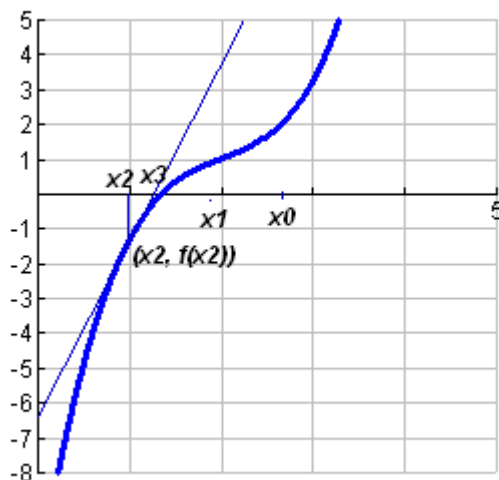
Use this x -intercept as the next approximation (x_1) for the zero of the function. Use it again to get x_2 , etc.

This is the iterative formula to use in a program.

Second graphic iteration:



Third graphic iteration:



Notice how the successive x -values (eventually) approach the actual zero (x -intercept) of the function.

Putting the formula (*) on the previous page $\left(x = x_0 - \frac{f(x_0)}{f'(x_0)} \right)$ into words:

“The *next* approximation (*x*-value) is the *current* *x*-value diminished by the value of the function at the *current* *x*-value divided by the derivative of the function evaluated at the *current* *x*-value.” This is called “one iteration of Newton’s Method”.

Then we let this new *x*-value become the current *x*-value, and repeat the process over and over, with each iteration generating another new *x*-value.

As this is such a repetitive process, it is ideal for implementation as a program. **NEWTON()** is a program which carries this out this iterative process until the point $(x_n, f(x_n))$ on the graph is less than 0.00001 away from the *x*-axis. (i.e. the value of the function is less than 0.00001 from zero.), You may either type this program, or download it from someone who already has it. Like **Bisect()**, it assumes that the function is stored in **y1**, and that you have some initial guess that is reasonably close to the actual zero you are searching for.

<pre> Newton() Prgm Input "guess",g g/(1.)→g Lbl a g-y1(g)/(nDeriv(y1(x),x) x=g) →n Disp n Pause If abs(g-n)<1. E - 005 Goto b n→g Goto a Lbl b Disp "done" EndPrgm </pre>	<p>Generate next guess, call it n</p> <p>Is next guess, n, sufficiently close to previous guess, g? If it is, exit loop.</p> <p>Next guess, n, becomes current guess, g.</p>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Run **Newton()** on the same function $f(x) = x^3 - 2x - 1 = 0$. **Newton()** takes only 6 iterations to find the zero, as compared to 17 iterations for **bisect()**.

Like the other iterative methods, Newton’s method does have its drawbacks. Sometimes the successive approximations actually diverge (get further away from the *x*-intercept of the graph). In rare instances, a tangent may be parallel to the *x*-axis, in which case there is no *x*-intercept for this tangent and the next approximation does not exist. You will examine these possibilities in exercise 4.

Tips for Sending and Receiving programs

First, make sure that the link cable is *carefully and gently, yet firmly, inserted* into the holes on the bottom edge of both calculators. If the cable is not properly inserted in both calculators, you will receive an error message. (See hints at bottom of page)

TIPS:

- Be sure that the link cable plugs are inserted completely. Be cautious, but firm. See below to recover from this error.
- **Sender:** be sure that the receiver has completed all of their 3 steps before you press **[ENTER]** in step 5. *The “receiver” must be waiting to receive before you send.* See below to recover from this error.
- **Receiver:** be sure you complete your 3 steps before the sender completes step 5. *You must be ready to receive before the “sender” can send.*
- If you receive the error message: **Link Transmission:** Press **[ESC]** twice on each calculator to cancel the transmission and return to the **HOME** screen. Check the transmission cable – it is probably not firmly inserted into one or both calculators. Or perhaps the sender completed all 5 steps before the receiver completed all 3 of their steps. It is also possible that the transmission cable is faulty. Contact TI (1-800-TI-CARES), or [email ticares@ti.com](mailto:ticares@ti.com). The instructor should have some extras that you may use in class.

Be sure the link cable is firmly inserted into both calculators

To SEND a program:

1. Press **[2nd] - [Var-Link]** (2nd key above **[ENTER]**). A list of the variables stored in your calculator will appear.
2. Use the cursor to move down to the name of the item you wish to send.
3. Press **[F4]:** $\sqrt{\quad}$. (This selects the item in preparation to send it.) You may select as many items as you wish to send.
4. Verify that the person with the receiving calculator has completed their 3 steps. After they have completed their steps 1 – 3, the message:
5. **Var-Link: Waiting to receive** should appear in the **STATUS LINE** of the receiver’s calculator.
6. Press **[F3]**, choose **Choice 1**, and **[ENTER]**. This sends the program to the other calculator.

To RECEIVE a Program:

1. Press **[2nd] – [Var-Link]** (2nd key above **[ENTER]**)
2. Press **[F3] : (LINK)**
3. Choose **choice 2: Receive** and press **[ENTER]**.
Look at the **STATUS LINE** at the bottom of the screen. The message:
VAR-LINK: WAITING TO RECEIVE should be displayed.

When the SENDER completes their step 5, you will see indication almost immediately that the program or script has been successfully sent.

If an error message appears instead, see below.

If an error occurs, on *both* calculators press one of these:
[ESC] *or* **[2nd] - [QUIT]** *or* **“[BREAK]”** (actually the **[ON]** key)
and start over

1. Run your **SqRoot()** program for the following:

a) Find the square root of 14, with a initial guesses of: 7, 6, 5, 4, and 3.8. Record the final results and the number of iterations (estimates AFTER the initial guess) it took to get the result.

Guess	7	6	5	4	3.8
Final Result (6 dec. places)					
Number of iterations to get final result					

2. Use the **bisect()** program to solve each of the following problems. Count the iterations required for the program to end. Record the indicated observations. You may use the table feature to see appropriate lower and upper bounds.

A. Find the smallest positive zero of $f(x) = 2^x - 3x^3 + \sin(x)$.

Find $f(0)$ _____ Find $f(1)$ _____. Since these have opposite signs, use them as your Lower Bound and Upper Bound, respectively.

Lower bound 0 Upper bound 1 The Zero _____ Number of Iterations required _____ Evaluate the actual *y coord.* at the zero: _____

B. Solve for the smallest positive value of x : $2x^4 - 4^x = \cos(x)$. Be sure the calculator is in radian mode.

To solve this equation using **bisect()**, you must create a function that has as its zeros the solutions of the equation. HINT: Move all terms of the equation to one side, with zero on the other. Count the iterations it takes to end the program. Record these observations:

Complete the table:

x	0	1	2	3
$f(x)$				

Lower bound _____ Upper bound _____ The Zero _____ $f(\text{the zero})$ _____ Number of Iterations required : _____

3. Now use the **Newton()** program to solve the same problems. Record your observations.

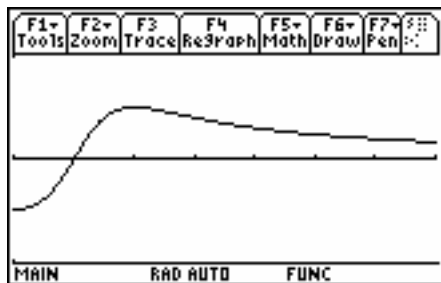
A. Find a the smallest positive zero of $f(x) = 2^x - 3x^3 + \sin(x)$. Count how many iterations it takes to end the program. Use the lower bound you used in problem 2A.

Initial Guess (lower bound in 2a) 0 The Zero _____ $f(\text{the zero})$ _____ Number of Iterations required : _____

B. Solve for the smallest positive value of x : $2x^4 - 4^x = \cos(x)$. Count how many iterations it takes to end the program. Use the lower bound you used in problem 2B

Initial Guess (lower bound in 2B) _____ The Zero _____ $f(\text{the zero})$ _____ Number of Iterations required : _____

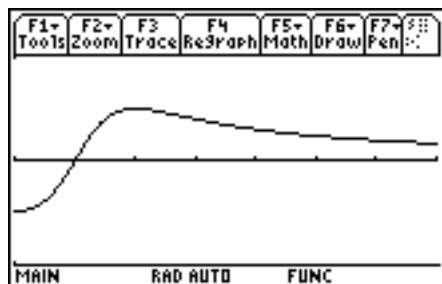
4. There are instances when Newton's method does not work, due to graph and the initial guess used. Consider the graph at the left, below. Notice the window coordinates shown!



Window: $[1, 8]_1 \times [-1, 1]_1$

A) Draw a horizontal tangent to this graph.

Estimate the x -value of this point. x -value at A _____
 Why will this value, when taken as an initial guess, cause Newton's Method to fail? Refer to the equation $x = x_0 - \frac{f(x_0)}{f'(x_0)}$ and/or the graph at the left.



Window: $[1, 8]_1 \times [-1, 1]_1$

B) There are points that also fail to yield an answer because the tangent lines' successive x -intercepts "move away from" the actual answer being sought. Locate such an initial guess (point) on the x -axis of the graph at the left. Label it B on the graph and estimate its x -value.

x -value at B _____

5. Modify the **Newton()** program so that execution will stop when two successive outputs agree to the eighth decimal place.

Only one line needs to be modified. Write it here: _____

6. EXTRA CREDIT: Modify the program **Bisect** or **Newton** to count the number of iterations performed to obtain the desired result. Three additional lines are necessary in each program. Write out the entire program.