Newton’s Method with TI-Nspire™ CAS

Forest W. Arnold

October 2018

1 Purpose

The objectives of this article are to:

1. explain what Newton's method is;
2. describe what Newton's method is used for;
3. describe short-comings of the method;
4. show how to implement Newton's method as a TI-Nspire CAS function; and
5. demonstrate using Newton's method in TI-Nspire CAS.

2 Description

Newton's method (also called the Newton-Raphson method) is an iterative technique for numerically approximating a solution (a root or zero) to a real-valued equation $f(x) = 0$. That is, it is a technique for finding an approximate value of an independent variable (usually x) that satisfies a given equation.

Determining the roots of equations is one of the most frequently-encountered problems in mathematics. Unfortunately, other than for a few types of equations, the roots of most functions can not be found using analytical methods. When this is the case, numerical methods such as Newton's method are used to find approximate values.

Iterative methods work by calculating a new value from a previous value, then using the new value to calculate yet another new value. This process (iteration) continues until some stopping criteria is met. An initial value is required for the iterative process to begin. Initial values are usually determined by choosing a value that is ”close to” the desired value, which for Newton's method, is a root of an equation. Stopping criteria for the process is usually a specified precision, which is a measure of how ”close” the calculated value (approximation) is to the actual (unknown) value.

3 Derivation of the Method

There are several ways of deriving Newton's method. Two of them are:

- Graphically - this is the derivation most often presented in a calculus course;
- analytically using Taylor series.

3.1 Graphical Derivation

Suppose you want to approximate the root, r, of a function $f(x)$. This means that $f(r) = 0$. Start by choosing a value x_0 and calculate $f(x_0)$ which yields the coordinates of the point $(x_0, f(x_0))$. Next, calculate the line tangent to the point $(x_0, f(x_0))$

and determine the x coordinate where the tangent line crosses the x axis. This new x coordinate is closer to the root r, so set $x1 = x$ and repeat the two calculations. Continuing these steps results in a sequence $x_0, x_1, x_2, \cdots, x_{n-1}, xn, \cdots$ with each succeeding x value closer to r. Figure 1 illustrates the first few steps in the process. Looking at

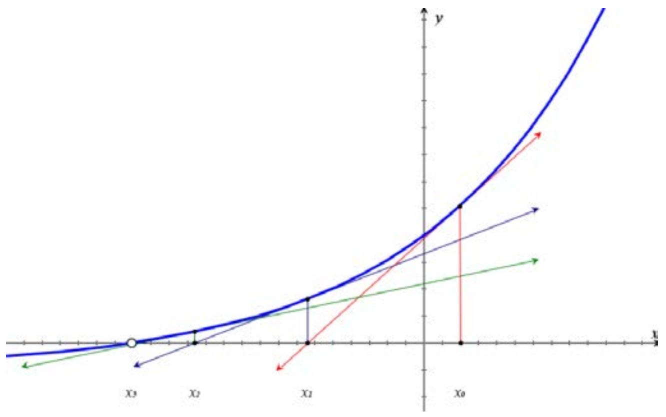


Figure 1: Newton's Method

Figure 1 shows how Newton's method works, but doesn't show how the tangent line and intersection calculations are done. A formula that models the process is needed. Here is how that formula is constructed for each x_k:

1. Given a value x_k, the coordinates of the point $(x_k, f(x_k))$ are found by evaluating $y = f(x_k)$.

2. The slope of the tangent line through the point $(x_k, f(x_k))$ is equal to the derivative of $f(x)$ at the point $(x_k, f(x_k))$: $m = f'(x_k)$.

3. The point-slope formula for a line with slope m through a point (x, y) is $y - y_0 = m(x - x_0)$. For $x = x_k, y_0 = f(x_k)$, and $m = f'(x_k)$, the equation of the line tangent to the point $(x_k, f(x_k))$ is $y - f(x_k) = f'(x_k)(x - x_k)$.

4. When the tangent line intersects the x axis, y equals 0, so the equation of the tangent line when $y = 0$ becomes $0 - f(x_k) = f'(x_k)(x - x_k)$. Expanding this equation and solving for x yields the equation

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

This simple formula is the mathematical model for Newton's method. It enables finding a new x value given a previous x value and the derivative of the function at the previous x value, provided that derivative is not equal to 0.

3.2 Taylor Series Derivation

The first Taylor polynomial for a function $f(x)$ expanded about a point x_k and evaluated at $x = r$, where r is a zero of $f(x)$ is

$$f(r) = f(x_k) + (r - x_k)f'(x_k) + \frac{(r - x_k)^2}{2} f''(\xi(r))$$

Assuming $|r - x_k|$ is small, then $(r - x_k)^2$ is even smaller. Dropping this small term and replacing $f(r)$ with its value of 0 yields the approximation

$$0 \approx f(x_k) + (r - x_k)f'(x_k)$$

Solving this approximation for r results in

$$r \approx x_k - \frac{f(x_k)}{f'(x_k)} = x_{k+1}$$

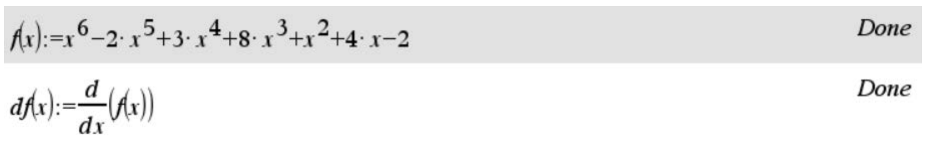
where x_{k+1} is a new approximation to the root r.

Derivation of Newton's method either graphically or analytically with Taylor series results in the same formula. The Taylor series derivation has one advantage over the graphical derivation: the dropped term $\frac{(r-x_k)^2}{2} f''(\xi(r))$, where $\xi(r)$ is between r and x_k, is the error when r is approximated by x_k. Even though the value of $\xi(r)$ is unknown (since it is somewhere between r and x_k), the dropped term can be used to find the maximum value of the error in the approximation.

4 An Interactive Example of Newton's Method

To see how Newton's method works, the two real roots (four roots are complex) of the polynomial $x^6 - 2 * x^5 + 3 * x^4 + 8 * x^3 + x^2 + 4 * x - 2$ can be found by interactively executing each iteration of Newton's method in a calculator application.

First, define functions for the polynomial and its first derivative:



$f(x):=x^6-2\cdot x^5+3\cdot x^4+8\cdot x^3+x^2+4\cdot x-2$ *Done*

$df(x):=\frac{d}{dx}(f(x))$ *Done*

Next, graph the function to determine the iteration start value, x_0. Figure 2 shows the graph.

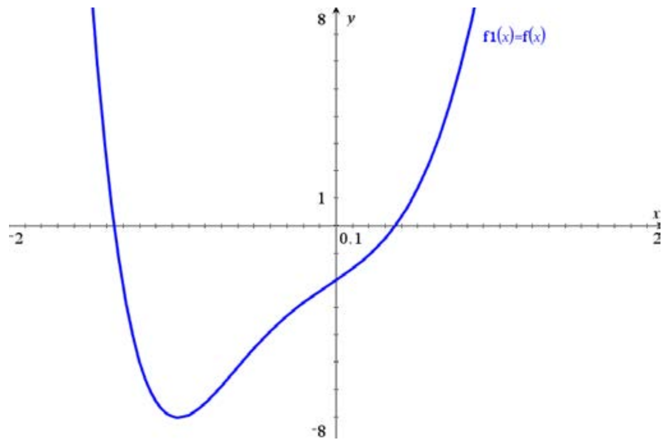


Figure 2: Graph of $f(x) = x^6 - 2*x^5 + 3*x^4 + 8*x^3 + x^2 + 4*x - 2$

Examine the graph to choose a value near the positive real root of the function. A value of 0.3 is near the positive root, so let $x_0 = 0.3$:

```
x0:=0.3                                                    0.3
```

To begin iterating, calculate x_1 using the formula for Newton's method with the value of x equal to x_0:

$$x1:=x0-\frac{f(x0)}{df(x0)} \qquad 0.367520569769$$

Calculate succeeding values the same way x_1 was calculated. Stop when a new value is the same as the preceding value:

$$x2:=x1-\frac{f(x1)}{df(x1)} \qquad 0.362055421603$$

$$x3:=x2-\frac{f(x2)}{df(x2)} \qquad 0.362014156978$$

$$x4:=x3-\frac{f(x3)}{df(x3)} \qquad 0.36201415465$$

$$x5:=x4-\frac{f(x4)}{df(x4)} \qquad 0.36201415465$$

The approximations from the fourth and fifth iterations are equal, so the approximation from the fourth iteration is accurate to within 12 significant digits.

As this example shows, when the initial approximation is near a root, Newton's method converges and does so after only a few iterations.

5 Drawbacks of Newton's Method

Even though Newton's method is an efficient numerical method for approximating roots, there are several ways it can either fail to converge, converge slowly, or converge to the wrong root:

1. if $f'(x)$ becomes equal to zero during an iterative step, the method fails since division by 0 is never valid;
2. if $f'(x)$ is near zero at any step, convergence may fail or be slow;
3. if two roots are near each other, the method may converge to the "wrong" root instead of the desired root; and
4. on rare occasions, depending on the function and the starting value, the method may get "stuck" in a cycle and alternate between two values forever.

When Newton's method is implemented in software as a function, these conditions need to be taken into account.

6 Newton's Method as a TI-Nspire Function

Executing Newton's method interactively is instructive, but is not a practical technique. Instead, creating a TI-Nspire CAS function that can be easily used to find roots of many different equations without a lot of typing is a better approach. Coding a function requires determining

1. The input arguments the function needs to perform its task;
2. the value(s) the function computes and returns;
3. exceptional conditions the function needs to accomodate; and
4. the steps the function needs to take.

6.1 Required Input Arguments

The required input arguments are:

1. A reference to the external function that returns the value of the function for a given x. The method needs to call this function internally. In TI-Nspire, references to functions are made using the name of the function and the function is dereferenced using the indirection operator, `#`.

2. A reference to the external function that returns the value of the first derivative of the function for a given x.
3. An initial value of x to start the iteration.
4. A tolerance value specifying the desired precision of the approximation. This value indicates how ”close” the approximation should be to the actual root. A good tolerance value for most numerical methods is $1.0e^{-11}$. The method ends when the absolute value of an approximation is less than or equal to the tolerance value.
5. A value specifying the maximum number of iterations the method should perform. This value is needed to prevent an ”endless loop” in cases where the method fails to converge to a root.
6. An optional ”flag” variable to turn diagnostic output on or off. When the value of this variable is `true`, the method displays information about what the method is doing during its processing.

6.2 Output of the Function

The function returns the value of the approximation if the method succeeds and returns the TI-Nspire symbol `undef` if the function fails.

6.3 Algorithm

An algorithm is a ”pseudo-code” description of a function or program. It describes the steps the code for the function or program follows. The alogorithm for Newton’s method is:

Newton’s Method Algorithm
Given a continuous function $f(x)$, its derivative $f'(x)$, an estimate $x0$ of a root of $f(x)$, an error tolerance value tol, and a maximum number of iterations $maxiter$, determine an approximation to the root of $f(x)$.

```
set $xold = x0$
set $numiter = 1$
while $numiter <= maxiter$
    set $xnew = xold - \frac{f(xold)}{f'(xold)}$
    if $|xnew - xold| < tol$ return $xnew$
    set $xold = xnew$
    set $numiter = numiter + 1$
endwhile
return undef
```

Pseudo-code algorithm descriptions are similar to TI-Nspire’s programming language, which makes it easy to implement the algorithms with the program editor.

6.4 Creating a Function with the Program Editor

TI-Nspire programs and functions are created with TI-Nspire's program editor. The TI-Nspire Programming Editor Guide has detailed descriptions of how to add a program editor to a document and create a program or function.

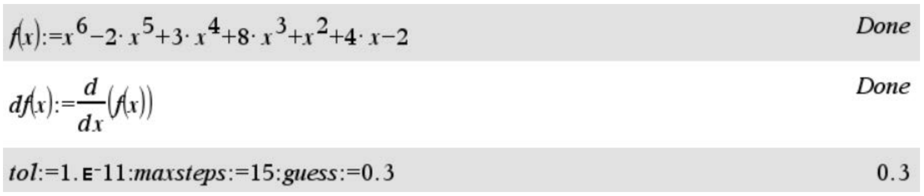
To create the function, open a program editor application, name the function `newtons` with type `function` and library access `libpub`. Type the following program statements in the editor page:

```
Define newtons(fname,dfname,x0,tol,maxiter,showsteps)=
Func
© newtons(fname,dfname,x0,tol,maxiter,showsteps) - find roots
© fname - the (string) name of the function
© dfname - the (string) name of the derivative of the function
© x0 - a value near the desired root of the function
© tol - tolerance value for desired precision
© maxiter - max number of iterations before stopping
© showsteps - display diagnostic or debug output at each step
©====================================================================
     Local xold,xnew,fval,deriv,numiter

     xold:=x0
     numiter:=1
     While numiter<=maxiter
         fval:=#fname(xold)            © use indirection to find f(x)
         deriv:=#dfname(xold)          © use indirection to find f'(x)
         If abs(deriv)<=1.@E-12 Then
             Disp "Warning: f'(x) near zero at iteration ",numiter
             Return undef
         EndIf
         xnew:=xold-fval/deriv
         If abs(xnew-xold)<tol Then
             If showsteps Then
                 Disp "Tolerance met at iteration ",numiter
             EndIf
             Return xnew
         EndIf
         If showsteps Then
             Disp "> at iteration ",numiter," xnew = ",xnew
         EndIf
         xold:=xnew
         numiter:=numiter+1
     EndWhile
```

```
    If showsteps Then
        Disp "Method failed to converge after ",maxiter," steps!"
    EndIf
    Return undef
EndFunc
```

When you finish typing the function, make sure you use the menu item `Check Syntax & Store` to save its definition. Next, test the function in a calculator page in the same document to make sure it is working correctly. Use $f(x)$ from the interactive example to compare the output of the function with the output of the example. Here are the commands to define values for the test:

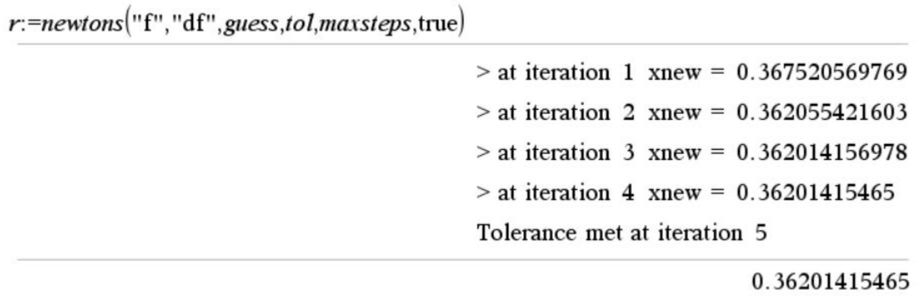


$f(x):=x^6-2\cdot x^5+3\cdot x^4+8\cdot x^3+x^2+4\cdot x-2$ *Done*

$df(x):=\frac{d}{dx}(f(x))$ *Done*

tol:=1.E-11:*maxsteps*:=15:*guess*:=0.3 0.3

And here is the function call and the output from the function:



r:=*newtons*("f","df",*guess*,*tol*,*maxsteps*,true)

```
> at iteration 1  xnew = 0.367520569769
> at iteration 2  xnew = 0.362055421603
> at iteration 3  xnew = 0.362014156978
> at iteration 4  xnew = 0.36201415465
Tolerance met at iteration 5
```

0.36201415465

Once you are confident the function is working correctly, save the document containing its definition in TI-Nspire's `mylib` folder, then open the library pane and select the `Refresh Libraries` button to make the function available to new documents.

7 Examples of Using Newton's Method

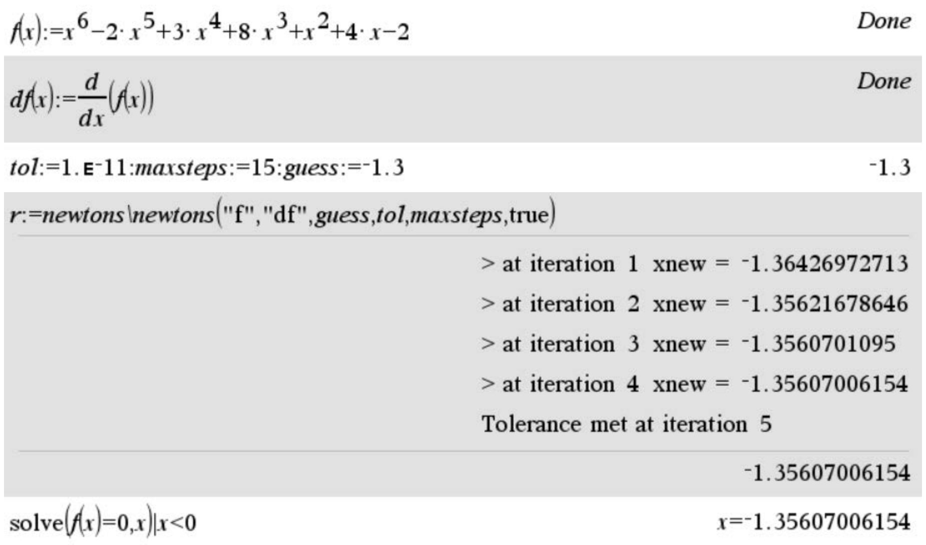
Each of the following examples is executed in a separate document and the `newtons()` function is added to the document from the library pane.

Note: The document containing the implementation of the `newtons()` function was saved as a library file named `newtons`. To add the function to a new document, open the library pane, select the `newtons` library, then double-click the name of the function. The function is added to a calculator page as `newtons\newtons()`.

7.1 Example 1 - Finding a Root of a Function

Find the real negative root of the polynomial function $f(x) = x^6 - 2x^5 + 3x^4 + 8x^3 + x^2 + 4x - 2$ to 11 digits of precision.

This example continues the interactive example discussed earlier. Examining the graph of the function indicates the negative root is near $x = -1.3$. The commands to execute the `newtons()` method and compare the result with TI-Nspire CAS's `solve()` function are:



```
f(x):=x^6-2·x^5+3·x^4+8·x^3+x^2+4·x-2                              Done
df(x):=d/dx(f(x))                                                  Done
tol:=1.E-11:maxsteps:=15:guess:=-1.3                               -1.3
r:=newtons\newtons("f","df",guess,tol,maxsteps,true)
                               > at iteration 1  xnew = -1.36426972713
                               > at iteration 2  xnew = -1.35621678646
                               > at iteration 3  xnew = -1.3560701095
                               > at iteration 4  xnew = -1.35607006154
                               Tolerance met at iteration 5
                                                         -1.35607006154
solve(f(x)=0,x)|x<0                                    x=-1.35607006154
```

7.2 Example 2 - Finding Where Two Functions Intersect

Find the coordinates of the point where $\sin(x)$ and x^2 intersect when $x > 0$.

The point where these two functions intersect is the point where the two functions are equal; i.e., $\sin(x) = x^2$. To use Newton's method to find points where the functions intersect, rearrange the equation as the function $f(x) = \sin(x) - x^2$. Solving this equation for its roots yields the x coordinate(s) of the intersection point(s). The y coordinate(s) can then be found by substituting the x coordinate(s) in either of the equations.

The graph of the two functions along with the graph of $\sin(x) - x^2$ is shown in Figure 3.

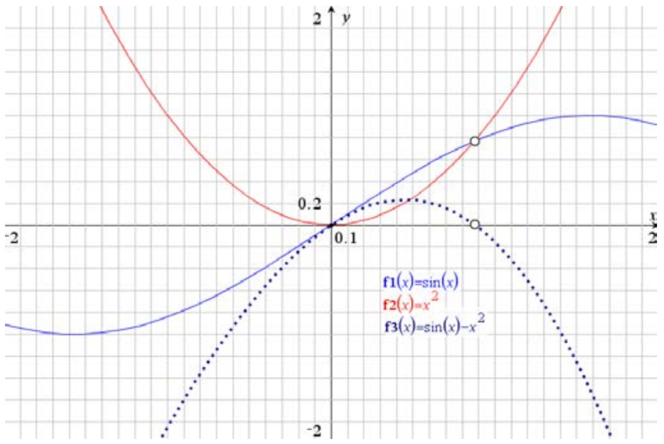
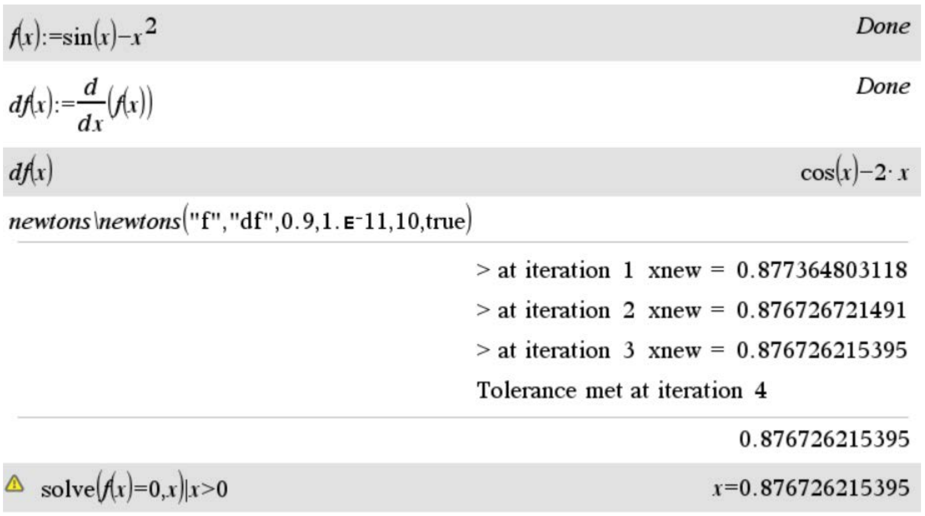


Figure 3: Graph of $\sin(x)$, x^2, and $\sin(x) - x^2$

From the graph, the positive x coordinate of the point where the two functions intersect is near $x = 0.9$. Use Newton's method to find the value to within $1.e^{-}11$ and compare the result with TI-Nspire's `solve()` function as follows:



$f(x):=\sin(x)-x^2$ Done

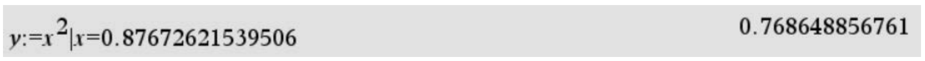
$df(x):=\frac{d}{dx}(f(x))$ Done

$df(x)$ $\cos(x)-2\cdot x$

```
newtons\newtons("f","df",0.9,1.E-11,10,true)
                    > at iteration 1  xnew = 0.877364803118
                    > at iteration 2  xnew = 0.876726721491
                    > at iteration 3  xnew = 0.876726215395
                    Tolerance met at iteration 4
                                             0.876726215395
```

⚠ $\text{solve}(f(x)=0,x)|x>0$ $x=0.876726215395$

The y coordinate of the intersect point is found by substituting the value of the root in either of the two equations using TI-Nspire CAS's constraint operator ($|$):



$y:=x^2|x=0.87672621539506$ 0.768648856761

7.3 Example 3 - Finding Critical Points of a Function

Find the first two critical points of the equation $\ln(x)\sin(x)$.

Critical points, also called *stationary points* in physics, are points where the first derivative of a function is equal to zero. Critical points are points where functions attain maximum or minimum values or change concavity. Thus the problem of determining maximum and minimum values of a function (if any) requires finding the critical points of the function. Newton's method can be used to find critical points by finding the roots of the equation $\frac{d}{dx}f(x) = 0$.

To solve this problem, first define the function and its first derivative in a calculator page:

$f1(x):=\ln(x)\cdot\sin(x)$	*Done*
$df1(x):=\frac{d}{dx}(f1(x))$	*Done*
$df1(x)$	$\ln(x)\cdot\cos(x)+\frac{\sin(x)}{x}$

The graph of the function and its first derivative is:

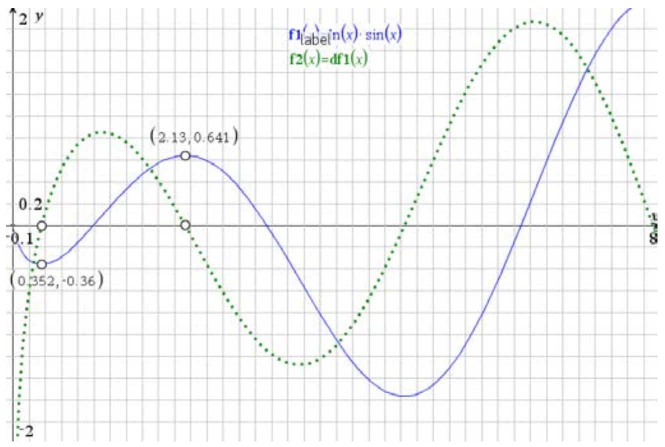
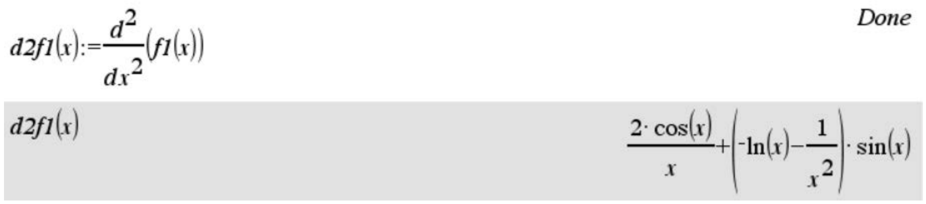


Figure 4: Graph of $\ln(x)\cdot\sin(x)$ and Its First Derivative

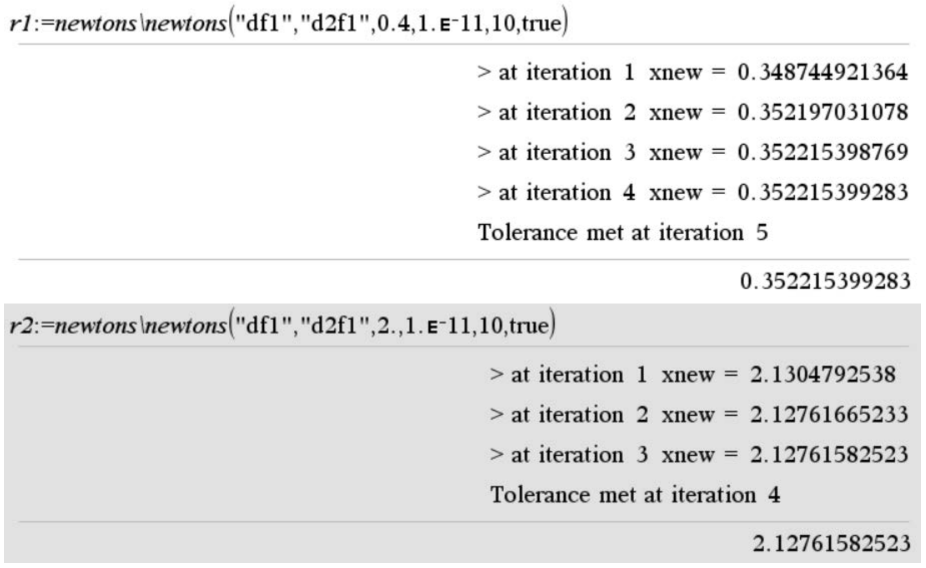
Examining the graph indicates that the first critical point (where the function has a minimum value) is near $x = 0.4$ and the second (where the function has a maximum value) is near $x = 2.0$. Using Newton's method to find the roots of the first derivative near these points requires defining a function for the first derivative of f'(x) (the second derivative of f(x)):



$$d2f1(x):=\frac{d^2}{dx^2}(f1(x)) \qquad \textit{Done}$$

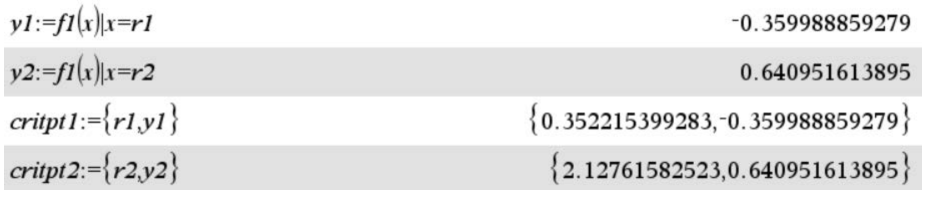
$$d2f1(x) \qquad \frac{2\cdot\cos(x)}{x}+\left({}^{-}\ln(x)-\frac{1}{x^2}\right)\cdot\sin(x)$$

The function `newtons()` can now be executed to find the x values of the first two critical points of $f(x)$:



```
r1:=newtons\newtons("df1","d2f1",0.4,1.E-11,10,true)
                        > at iteration 1  xnew = 0.348744921364
                        > at iteration 2  xnew = 0.352197031078
                        > at iteration 3  xnew = 0.352215398769
                        > at iteration 4  xnew = 0.352215399283
                        Tolerance met at iteration 5
                                                 0.352215399283
r2:=newtons\newtons("df1","d2f1",2.,1.E-11,10,true)
                        > at iteration 1  xnew = 2.1304792538
                        > at iteration 2  xnew = 2.12761665233
                        > at iteration 3  xnew = 2.12761582523
                        Tolerance met at iteration 4
                                                 2.12761582523
```

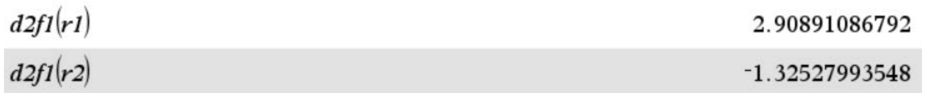
The values $x = r1$ and $x = r2$ can be used to find the y values of the critical points by substituting them in $f(x)$ and the (x,y) coordinates of the critical points can be placed in lists for later use:



```
y1:=f1(x)|x=r1                          -0.359988859279
y2:=f1(x)|x=r2                           0.640951613895
critpt1:={r1,y1}        {0.352215399283,-0.359988859279}
critpt2:={r2,y2}          {2.12761582523,0.640951613895}
```

It is clear from the graph in Figure 4 that the first critical point is a local minimum and that the second critical point is a local maximum. This can be determined *analytically* using the *second derivative test*. Recall from calculus that if the second derivative at a critical point is greater than zero, the critical point is a local minimum, and if the second derivative at a critical point is less than zero, the critical point is a local maximum. When the second derivative at a critical point is equal to zero or is undefined, the second derivative test fails to identify local minimums or maximums.

The second derivative test can be easily performed with TI-Nspire CAS:



```
d2f1(r1)                                   2.90891086792
d2f1(r2)                                  -1.32527993548
```

Evaluation of the second derivative for $r1$ is positive and for $r2$ is negative, verifying that the first critical point is a local minimum and the second critical point is a local maximum.

8 Summary

Newton's method is an efficient and accurate numerical method for approximating roots of functions. The method is easy to implement as a reusable library function with TI-Nspire's Program Editor. The method can be combined with TI-Nspire's built-in algebra and calculus commands to solve mathematical problems involving roots of functions.

References

[1] Burden, Richard L. and Faires, J. Douglas, *Numerical Analysis, 9th Ed.*, Brooks/Cole Cengage Learning, 2011, pp. 67-71

[2] Briggs, William, Cochran, Lyle., and Gillet, Bernard, *Calculus for Scientists and Engineers Early Transcendentals*, Pearson Education, Inc., 2013, pp. 302-309