Newton-raphson method example

Newton's method for thought to find the root of the equation is easiest to understand by example. At the very least, I learned more easily than an example. So, maybe you did, too. In this article I have collected some very teaching examples for the Newton-Raphson method and for what it does. You'll see it work well and fail amazingly. I show you the similarities and pictures you need to understand what's going on, and I give you a piece of python code so you can try all of it yourself. Before we dive into the example, let me mention that I have written a complete introduction to the Newton method itself, here on ComputingSkillSet.com, in my article The Newton Method Explains: Details, Pictures, Python Code. If at any step in one of the examples you feel you can benefit from seeing this particular step explanation, just jump there and come back afterwards. And another thing: The Newton method of finding functional roots is also called the Newton-Raphson method, so I'll use both of these names to be modified. Example 1: Calculating the square root of a positive number with the NewtonNewton method can be used to calculate the square root of a positive actual number. Sounds interesting, right? But how does this calculation algorithm help calculate square roots? Here's how:Let's say that we're interested in square root 2. Then, we see the function\[f(x)=x^2-2\]Why is this function? Because of the way the Newton method works is that it can help us find zero functions, if we already have a fabrix idea where zero is like that. To use this strategy, the above function is designed so that it has zero on, Yes indeed, a square root of two:\f(x)=0\quad\rightarrow\quad x^2-2=0\quad\rightarrow\quad x^2=2\]This means that if we use the Newton method to find zero (root) of this function, it will give you the At this point, it's clear, by the way, what to do, if you want a square root seven instead: You'll use \(f(x)=x^2-7\). But because we want to see the numbers out that many people recognize, let's stick to the square root of two. Here's an overview of what this function is and its derivative looks like: The Newton method tells you to start with an initial guess, \(x_0\), and then change it the following lines until you are satisfied with the result or the failed algorithm:\[x_{i+1}= x_i – \frac{f x_i(x_i)}this becomes\[x_{i+1}= x_i – \frac{x_i^2-2}{2x_i} \]and that's quite easy to do , even by hand, if we wish. Ok, first we pick the initial guess. We know (playing naive on purpose here) that roots Two are somewhere between numbers 1 and 2, so let's choose \(x_0=1.5\), that's right in the middle. Then we get\[x_1= 1.5 – 1.5 - \frac{2.25-2}{3}= 1.5 - \frac{0.25}{3}=1.5 -0.08333333= 1.41666666 \]So this is the first step. One thing we need to note is the relative difference in \(x_i\) from one step to the next, namely:\frac{|x_{i+1}- x_i|} {|x_i|} = \frac{|f(x_i)|} {|f'(x_i)x_i|} \]For certain examples and steps we have,[\frac{|f(x_0)|} {|f'(x_0)x_0|} =0.05555555 \]Ok, next step. This time, doing calculations without a calculator isn't very fun anymore, but I'll still note the numbers as if we were going to do it that way:\[x_2= 1.41666666 – \frac{1.416666666^2-2}{2.22.3833333}= 1.416666666 – 0.00245098= 1.414215686 \]To those of you who know the answer with heart, this should look pretty good. To maintain our score, let's check the relative differences. It is\\frac{|f(x_1)|} {|f'(x_1)x_1|} =0.0017301 \]On to the next step. We got\x_3= 1.414215686275 - \frac{1.414215686275^2-2}{2.828427137255}= 1.2}{2.828427137255}= 1.1.414215686275 – 0.000002123900= 1.414213562375 \]You definitely realize that I keep adding more digits to those numbers, and that's because it is necessary. The result is getting closer to the true root of our equation really quickly now. Let's take a look at the relative differences for this step:\\frac{|f(x_2)|} {|f'(x_2)x_2|} =0.0000015 \]So, because the next step should be (and will) smaller than this one, we can estimate that our results are already accurate to 5 digits or more. Another step, ok? Here we go:\[x_3= 1.414213562375 - \frac{1.414213562375^2-2}{2.82842712475}= 1.414213562375 – 0.0000000000002= 1.414213562373 \]Only minor changes. In fact, the relative difference is \(0.00000000001\) and we are completed. This works well and is quite quick, because we are close to the roots and the function behaves well. To describe this, here is an ancient plot built to go these steps. It's all very close and hard to see anything:Function is plotted in blue and hard to see under red tents. The red stars marked spies on the curve, where his steps were. Green Xs mark the same points on the x-axis, and the handsome lines of green lead to each other's stars. Even for certain options of this initial guess, fast convergence, it is different for other initial guesses. To get the idea, how different, we can plot the number of maintenance measures needed to achieve a certain level of relative accuracy, in our case (10^{-10}\). The result is the following figures:This red dot shows fast concentration areas, where they are low, and slow concentration, where they are high. In general, a small slope of the function in question at certain initial guesses leads to a slower concentration. Instances The original is very inappropriate, since there is a vanishing slope and so Newton's algorithm fails. In summary, we have seen that the Newton method can be to get the probable values for the square root of a positive real number in a simple way. Example 2: Calculating the cubic root of a positive number with the next example NewtonThis method is the same as the first, but will be a little bit more annoying to do by hand. We will use the Newton-Raphson method to calculate the cubic root of number 2. This is an unusual number like a square root of two, but it's easy enough to check out a pocket calculator that can do cubic roots. So here we go. The function we define for this purpose is (can you guess after reading the first example?):\ [f(x)=x^3-2\] This function has a root (single) property in \(x=\sqrt[3]{2}\), which is what we want to know and what method Newton will help us find. The function and appearance of its derivatives like this:To start a search, let's choose the initial guess, say \(x_0 = 1.8\). With the general formula\[x_{i+1}= x_i – \frac{f(x_i)}{f'(x_i)} \]we found for our solid-root generation function\[x_{i+1}= x_i – \frac{x_i^3-2}{3x_i^2} \]Using this to \(x_0 = 1.8\), we arrived at x_1 = 1.405761316872\) and relative differences \(0.219021490626\). Repeat procedures (for more information, see example 1 above), we get \(x_2 = 1.274527978340\) with relative differences \(0.093353926415\), \(x_3 =1.260087815373\) with relative differences \(0.011329812458\), \(x_4 = 1.259921071964\) with relative difference \(0.000132326816\), \(x_5 = 1.259921049895\) with relative differences \(0.00000017517\), and \(x_6 = 1.259921049895\) with relative differences \(0.0000000000000000\). Yes, that's right – the last step doesn't change anything in the digits shown here. So that is our result:\sqrt[3]{2}=1.259921049895\]In the picture, these measures can once again be described by the estuary that suits it:In this case, at least the first towel measures and the keys are clearly distinguished from the other, before the plot becomes crowded near real value. Again, we can ask about reliance on the number of necessary steps on the initial guess. Testing this and plotting these numbers across the range plotted in the overview, we got:Low areas in the red-dot curve provided a good solution after a few low auction steps, while high areas took longer. The point to avoid again the origin, in which the slope of our function vanishes and the algorithm of the Newton-Raphson method stops. Example 3: Calculating any positive root number with the NewtonThis method example also deals with calculating the roots, but I want to keep this brief. You already got an idea from the first two with concrete numbers and illustrations. In this example, I want to simply write a general formula for calculating the roots number \(a\). To find a number that is root (\(n>1\)) from the actual number positive \(a\), which function do we need to use in the Newton method? We use\[f(x)=x^n-a\]because this is zero, where \(x=\sqrt[n]{a}\). Now, remember the Newton-Raphson broadcast move, namely\[x_{i+1}= x_i – \frac{f(x_i)}{f'(x_i)} \]and replace to \(f\) and \(f'\) accredited. That brings us to\[x_{i+1}= x_i – \frac{x_i^n-a}{n x_i^{n-1}} \]In the same way, we can clearly build a iteration step in terms of the functions we want to investigate. This works, provided that we can write a functional form for its functions and derivatives. In the next few instances we will see and experience a variety of ways, in which newton methods can and will fail. Example 4: Newton method fails when there is no root in my article Newton Method Explains: Details, Pictures, Python Code, I mentioned some cases where the Newton method can or will fail. In the following, I presented several examples to exactly those cases. The first is that the function, for which we try to find the roots, actually has nothing. Now you might say that this is an artificial state, because we usually know whether the functionality we investigate has root or not. Although this is true for something like \(f(x)=x^2\), it can indeed be more complicated in general. Think about the following circumstances: You tell me about some interesting phenomena. We are thinking about similarities together that are supposed to describe this phenomenon. But our similarities are not algebraic to start with, so we don't know what the solution looks like, we just know that it should be a single variable function, which the most important part is zero (root). So in the first step, we need to calculate the functionality we want to use the Newton method. And then it can happen that there is no roots for this function. Ok, enough with the mind experiment, let's take a look at the example. Above, we have used the function\[f(x)=x^2-2\]to calculate the square roots of two, because we know that its roots are the numbers we want. If we move this function by adding number four to it, we get\[f(x)=x^2+2\]and no (real) root yet. So, let's try the Newton method on this and see what happens. Its functions and appearance of derivatives like this:Here we can clearly see that the (blue) function does not even come near zero. Its derivative is problematic in the same fashion as above, that is, there is zero value for the slope on the origin. I started the Newton method at \(x_0=1.8\) and gave it 10 steps to try and find a solution (remember that this is enough in the example of finding the above square root), but (obviously), an algorithm keep wandering around:To see immediately that simply using more steps doesn't (because it can't) help, here is the same plot again, but this time Possible steps, i.e. the maximum 100 lights:So, there are 100 noticeables in this picture (or maybe not everything actually appears between \(x=-2\) and \(x=2\). You can see a lot of them, that's pretty good. So, if your Newton-Raphson auction doesn't get together and you don't know why, this case is a possibility. Example 5: Newton's method of running to the functional asymptotic regionAnother is likely a failure of the Newton method to occur, when there is an asymptotic region in functions, for example, the function falls from the monotony toward positive infinity, but does not reach zero. What? Let's take a look at the concrete example. We will use the function\[f(x)=x \exp(-x)\] which has such asymptotic region towards positive infinity. Here it is plotted along with its derivatives:We can clearly see that the function actually has the roots, which the Newton-Raphson method can also find, but only if it starts at the appropriate point. Our selection \(x_0=1.8\), for example is located near the maximum function, more precisely, on the maximum right. If the search for root with the Newton method starts there, the following happens:We observe that the slope on the maximum right is always negative. So, once the method starts there, \(x_i\) can only move to larger and larger numbers and their sequence is actually different great. If, on the other hand, a search is started on the maximum left, say on \(x_0=0.5\), the method of finding the roots after 6 steps. Here's a graph with a lease path presented by red stars on the curve and the green Xs on the x:I axis should mention here that chatting towards zero solutions (as is the case here) is slightly problematic when defined through relative differences. Once the level of root accuracy is such that it appears as zero in computer memory, division by zero encountered. However, meaningful values are produced on the way to this failure, so we can still feel the content about avoiding the asymptotic trap of this function. For this case, I showed the third type of plot again (as in the first instance), where the number of cheering steps needed for the solution was added to the overview. Here is:The red dot shows the value (clearly discriminatory) for the required number of broadcasts, if the Newton method is started at this value to \(x_0\). Keep in mind that there are a maximum number of leopards in my code to avoid infinitive loops when the algorithm fails to gather and stop. In this plot I set that maximum number for 20.We can clearly see that the plateau at 20 appears at the maximum value of our functionality and is there for the rest exponent tail asymp;asymp;asymp;asimptotic To the left of the maximum \(f\), there is our usual item: less lengat closer to the root. To summarize this example: if you see your value for to run in one direction, perhaps that's the fault of the asymptotic region that your function shows there. Example 6: The Newton method swings between two foreverAnother regions the possibility for newton methods stuck in insane search patterns is a swing. In particular, in such cases curved procedures with turmoil jump between the two provinces, which slope the point to each other. Now what should it mean? Let's take a look at the example. The function that may occur that will show us this behavior is\[f(x)=-\cos\left(|x|+\frac{\pi}{4}\right) + 0.8\] whose first derivatives\[f'(x)=\sin\left(|x|+\frac{\frac{\frac{\frac{\frac{\frac{\pin}{4}\right)\\ \text{sgn}(x)\]This is still difficult to fix in your head, So let me show you the graphs of these two:The region interesting for our purposes is in the middle, where curvedness is positive. Any Newton search there will be trapped and can't go out, for example:What does it mean for newton method convergence areas for this function? There is a region in the middle, around the original, where the method failed. Beyond that it works most of the time, but fails, where one of the steps in the iteration leads to a sword trap in the centre:This may seem like an untyploded case because of functional construction with absolute value. However, absolute value is not uncommon as we will probably think or like. So, if you look at the eyes of the Newton method you swing (in a sign or otherwise), then something like the case indicated here might happen. Example 7: The Newton method of failing to root increases slower than square root is the root that the Newton method does not gather, but it differs from. Why? Because the slopes and functional curvity in the root produce measures of the inflation that go further from the roots, although the initial guess is chosen very close to it. The boundary behavior of the function in question for this occurs is square root, that is, if \(f(x)\) at the root increases slower than\f(x)=\sqrt{x}\then the Newton method stays away from the roots. Let's try this by investigating the solid-root function:\f(x)=\sqrt[3]{x}\In this case, Derivatives are\[f'(x)=\frac{1}{3 x^{2/3}}\]and the cooling step becomes\[x_{i+1}=x_i – 3x_i=-2x_i\;. \]Now, for this example we know that the root is located in \(x=0\). This means, looking at the steps, we see how x_i\) at least sign and grow in size at each step, that is, it moves away from the roots at zero. Nulliently, in this particular example, the initial guess will be multiplied by a negative factor of two at each step of iteration, growing geometry beyond all bounds. As a result, exercising this as a probable exercise can only help reveal the the maximum javelin that is lost, or produces a abundance of conjecty. Example 8: Newton Newton for arctangent functions of similar examples, but with the concentration region in the center, is the arctangent function:\[f(x)=\arctan(x)\]with derivatives\[f'(x)=\frac{1}{x^2+1}\In this case the supervision step becomes\[x_{i+1}=x_i – (x_i^2+1) \arctan(x_i)\x_i x_i)?? arctan(x_i).\x_i \]Now, for the choice \(x_0\) is pretty close to the original, we get a conversion pattern, for example, to \(x_0=1\):Anyway, if we start a little further from the origin, say at 1.5, the Newton method differs greatly:Here, the method of ying and different from the origin, resulting in a larger number of the computer can handle less than 20 steps. The boundary between the two regions (convergence vs. differences) is available by resolving the equation \(x_{i+1}=-x_i\), because the resulting sequence is subtitle in the mark. We found\[x_i – (x_i^2+1) \arctan(x_i)=-x_i\]and, facilitate the equation,\\arctan (x_i)=\frac{2x_i}{x_i^2+1}\]whose solution is approximately\19175). If the absolute value of the initial guess is greater than this amount, the Newton method is different, for the absolute value smaller than the initial guess, it accumulates. Example 9: Some roots to choose for the NewtonAmong method are all these examples for the use of newton methods to find functional roots, we haven't seen the most common yet. So far, we have a case with one root, no roots, but not some roots. We've seen that method fail, but sometimes, the problem is not that it fails, but it fails to find the solution we want it to find. Sounds complicated? It could be. Here's an example. For this purpose, polinomial is an ideal type: it is easy to build, can have several roots in a particular location, and derivatives are also easily available. Therefore, let's use\[f(x)=(x+2)(x+1.5)(x-0.5)(x-2)\]with the first derivatives\[f'(x)=(x+1.5)(x-0.5)(x-2)+(x+2)(x+2)(x-2)(x-2)(x-2)(x-2)(x-2)(x-2)(x-2)(x-2)(x-2)(x-2)(x+2)(x-2)(x-2)(x-2)(x-2)2)(x+2)(x-2)(x-2)(x-2)(x-2)(x+2)(x-2)(x-2)(x+2)(x-2)(x-2)(x-2)(x-2)(x-2)(x-2)(x0.5)(x-2)+(x+2)(x+1.5)(x-2)+(x+2)(x+1.5)(x-0.5)\]Both look like this :One can clearly see the four roots in function, and the derivative has three zero, in which the Newton method will stop , if they are beaten exactly. Moreover, there is no obvious problem. The concentration plot (which shows the number of transparent required to arrive at a constant solution as a red dot for each \(x_0\)) shows exactly what I mentioned: Nice convergion everywhere, except when hitting zero in the first derivatives. However, it is an interesting case for the following reasons: Let's try and calculate the roots in \(x=0.5\). And let's pretend that this function is actually unknown to us in detail (playing stupid to demonstration). We somehow figure that value \(x_0=1\) would be a good idea, so we started the Newton method run with this initial guess. Here's the resulting search with all tangents:Wow, this works well! We found the roots, and better, we found our targets! But what happens, if we don't get so and someone has suggested \(x_0=1.3\)? Here's what happened:Wow again, but in a different way! This search jumps to another root than closest to our starting point. And moving further to a more positive start value will take it to the next negative root in \(x=-2\). Then, just a little bit more, it will accumulate towards the most positive roots on \(x=2\). So, you see that a little difference in the initial guess can make a considerable difference in terms of where the Newton method goes with its search pattern. As a result, we should always try to get as close as the roots with our initial guess! Example 10: Fractals produced by the Newton method, indeed, there are fractals that can be generated using the Newton method. In short, the background is this: Although I keep stressing above in some places that we are looking for real roots, newton methods can be generalized to complex aircraft (that is, for complex numbers and functions). If the function is quite well behaved also for a complex argument, then we can also extend the validity of its derivatives from the actual axis. If this is too much information for you, don't worry. Imagine calculating a functional derivative with two real variables instead of one. Now, how and why does fractals connect to Newton's methods and vice versa? The answer lies in previous instances. There, we have seen that the Newton method can accumulate to the root of different functions of the initial guess of the x_0 neighbor\). So far, we only see the number of effects needed to arrive at a constant solution - any solution. But now imagine that we are coding each color \(x_0\), depending on the root \(f\) detour gathered. So, if there are four roots, as in previous instances, there will be four colors. And then, we run the Newton method, starting from every point on the two-dimensional grid of complex numbers. We observed two things: which root the method gathered and how many leanings were needed to get there. The first (which root) translates into the color of the starting point on the grid, and the second (how much iterations) can be used for the tense addition of that point color. I know this is a bit all at once, which is why I've written the whole article in addition to this one, just about Newton fractals. You can find it here: Newton Fractals Explained: Examples and Python Codes. Here's a gosa-like spying. A certain example is that we used above for example number 7:As you can see, this picture only uses color information and is not shady – I leave the latter for the in-depth articles linked in Example 11: Finding roots with greater diversity in polinomials with the NewtonAnother method of common examples related to the Newton method is the presence of various roots among the roots polinomial. So far, we have seen only simple roots in our example. At this point, I want to show you another with double roots. We will only reuse functions from example 7, but with a small difference:\[f(x)=(x+2)(x+1.5)^2 (x-0.5)(x-2)\Note that the second factor is square now, meaning that this polinomial root in \(x=-1.5\) is now double. The first derivative reflects this as well and becomes\[f'(x)=(x+1.5)^2(x-0.5)(x-2)+2 (x+2)(x+1.5)(x-0.5)(x-2)+(x++2)(x+1.5)^2(x-2)+(x+2)(x+1.5)^2(x-0.5)\Our functional graph and the appearance of the first derivatives are similar to those above example 7, but slightly different:Admittedly, hard to see the region around double roots, so we'll go straight to the search plot. Starting at \(x_0=-1\), we arrived at the desired solution:Here someone can see a better functional graph around the double root. We also see how kuhemah, star and Xs are moving toward a solution. What is unclear from this figure, but important and the reason for this example, is the speed of convergion towards multiple roots as opposed to a single root. For this, our usual plot with the number of iterations needed as a useful starting point function. Here it is: Apart from the occasional jitter around the first zero derivatives, there are two obviously different areas with respect to the number of iterations required. The region around the roots doubles in \(x=-1.5\) and everything, which we can call the region around the roots is easy. In particular, we see that around 30 ignorance is needed to arrive at constant results for multiple roots, while 5 or 6 are enough to get to a simple root with the same relative accuracy. The reason is the theory, which is that the Newton method usually accumulates quite well (quadratically), but not for various roots such as our double example here. Basically, you can modify the Newton-Raphson algorithm, but to do it right, you need to know the diversity of roots. So, in practice mostly doesn't help as we expect. Let's remember that if the Newton method of gathering unusually slowly, there may be multiple roots behind that behavior. Python Code to generate solutions and numeric figures for these examples for the NewtonFinally method, so you have a few concrete starting points to learn more and play around with this Newton method and these examples, here are the python codes I use to do the calculations and produce images. It comes like and without any warranty, so use it at your own risk. If you want to reuse or share this code elsewhere, you can also do so – please read my notes on my code example. allegedly, have fun!#/usr/bin/env python3 #-*- coding: utf-8 -*- Created on Tue Sep 25 05:52:47 2019 Newton-Raphson method code. Resolving methods a given preliminary guesswork as well as testing the entire x grid as the initial guesswork Produces three figures: the overall overview function of tangen to guess the initial efficiency selected from other initial guesses Includes some presets for functions, but easily extended @author: Andreas Krassnigg, ComputingSkillSet.com/about/ This work is licensed under creatives Attribution-ShareAlike 4.0 License More information: of matplotlib.pyplot imports as lumps of import plt as np # definition function to evaluate f and f' on x. Add yourself to need. # define its functions and derivatives to investigate: # Demo Def function f(x): valve = np.sin(x**2) - x**3 - 1 valve = 2*x*np.cos (x**2) - 3*x**2 returns (valves, Valves) # example 1: square root two def e1(x): valve = x**2 - 2 valves = 2*x returns (valves, valves) # example 2: root the two cube def e2(x): valve = x**3 - 2 valves = 3*x**2 back (valf, valve valf = x**2 valves) # example 3: seven-root def e3(x): valve = x**7 - 7 valves = 7*x**6 returns (valve, valve) # example 4: function without root def e4(x): valve = x **2 + 2 + 2 valves = 2*x returns (valve, valdf) # example 5: function with asymptotic region def e5(x): valve = x*np.exp(x) valve = -x*np.exp(x) return (valf) , valdf) # example 6: function with oscillation def e6(x): valf = -np.cos(np.absolute(x)+np.pi/4) + 0.8 valdf = np.sin(np.absolute(x)+np.pi/4)*np.sign(x) return (valf, valdf) # example 8: arctangent def e8(x): valf = np.arctan(x) valdf = 1/(x**2+1) return (valf, valdf) # example 9: function with many roots def e9(x): valf = (x+2)*(x+1.5)*(x-0.5)*(x-2) valdf = (x+1.5)*(x-0.5)*(x-2) + (x+2)*(x-0.5)*(x-2) +(x+2)*(x+1.5)*(x-2) + (x+2)*(x+1.5)*(x-0.5) return (valf, valdf) # example 11: function with many roots, one multiple def e11(x): valf = (x+2)*(x+1.5)**2*(x-0.5)*(x-2) valdf = (x+1.5)**2*(x-0.5)*(x-2) + 2*(x+2)*(x+1.5)*(x-0.5)*(x-2) +(x+2)*(x+1.5)**2*(x-2) + (x+2)*(x+1.5)**2*(x-0.5) return (valf , valdf) #defines the parameters to plot – adjust it as necessary for your function #defining left and right borders to plot # to plot the overview: interval_left = -2.1 interval_right 2.2.1 interval_down = -25 interval_up = 60 # for search plot: interval_left_search = -2.1 interval_right_search = -0.7 interval_down_search = -0.0.0.2 interval_up_search = 2 # set the number of points on the x axis to plot and guess the beginning of the num_x = 1000 # determine an initial guess you want to use to get the option solution 1 # the desired accuracy set and maximum number of iterations prec_goal = 1.e-10 nmax = 100 # the following defines the function to complete and plot. Jump to the end of the code to select the #definition x eye grid function for calculation and plotting function xvals = np.linspace (interval_left, num=num_x) # create a list of starting points for Newton's method for coils later in the code # we will be semua x nilai sebagai titik titik permulaan = xvals def solve_and_plot_newton (func): # mula berkomplot rajah 1: proses carian dengan khemah plt.figure() # plot fungsi f dan derivatif #fx, dfx = f(xvals) fx, dfx = func (xvals) plt.plot (xvals, fx, label='$f(x)$') # plot garis sifar untuk rujukan mudah plt.hlines(0, interval_left, interval_right) # menetapkan had untuk paksi x dan y dalam rajah # set x had plt.xlim ((interval_left_search, interval_right_search)) # menetapkan had y plt.ylim ((interval_down_search, interval_up_search)) #label paksi plt.xlabel() x, fontsize=16) # memulakan nilai untuk gelung lelaran reldiff = 1 xi = pilihx0 kaunter = 0 print ('Memulakan kaedah Newton di x0 =',dipilih) # memulakan gelung lelaran manakala reldiff > prec_goal dan kaunter < nmax : # do the necessary computations # get function value and derivative fxi , dfxi = func(xi) # compute relative difference reldiff = np.abs(fxi/dfxi/xi) # compute next xi x1 = xi - fxi/dfxi # print numbers for use in convergence tables (csv format) print('%i, %15.12f, %15.12f, %15.12f' % (counter+1, x1, fxi/dfxi, reldiff)) # plot the tangents, points and intersections for finding the next xi # plot xi on the axis plt.plot(xi,0,'gx', markersize=16) # plot a line from xi on the axis up to the curve plt.vlines(xi, 0, fxi, colors='green', linestyles='dashed') # plot star on the curve where x=xi plt.plot(xi,fxi,'r*', markersize=16) # compute y values for tangent plotting tangy = dfxi*xvals + fxi - dfxi*xi # plot the tangent plt.plot(xvals , tangy, 'r--') # plot next xi on the axis plt.plot(x1 ,0,'gx', markersize=16) # trade output to input for next iteration step xi = x1 # increase counter counter += 1 # print test output print(chosenx0,x1,counter,reldiff) # remember counter for later plotting counter_chosen = counter plt.savefig('newton-method-example-plot-search.jpg', bbox_inches='tight') #plt.savefig('newton-method-example-plot-demo.jpg', bbox_inches='tight', dpi=300) # close figure 1 plt.close() # begin plotting figure 2: overview of function and its derivative plt.figure() # plot the function f and its derivative #fx, dfx = f(xvals) fx, dfx = func(xvals) plt.plot(xvals, fx, label='$f(x)$') plt.plot(xvals, dfx, label=$f'(x)$) # plot the zero line for easy reference plt.hlines(0, interval_left, interval_right) #label the axes plt.xlabel(x, fontsize=16) # plt.ylabel($y(x)$, fontsize=16) # menetapkan had untuk x dan y paksi dalam rajah # set x had plt.xlim ((interval_left , interval_right)) # kita tidak menetapkan had y plt.ylim ((interval_down, interval_up)) # tambah lagenda ke plot plt.legenda() # simpan angka ke fail output 1: gambaran keseluruhan fungsi dan derivatif plt.savefig ('newton-method-example-plot-plot-gambaran keseluruhan.jpg', bbox_inches='ketat') # menjalankan ploting rajah 3: gambaran keseluruhan ditambah # gelung ke atas semua titik permulaan yang ditakrifkan list of points for x0 in the point list: #initialize values gelung legaji reldiff = 1 xi = x0 kaunter = 0 # memulakan gelung lengkungan manakala reldiff > prec_goal dan kaunter < nmax: # mendapatkan bilangan lelaian yang diperlukan pada x0 # compute perbezaan relatif fxi, dfxi = func(xi) reldiff = np.abs (fxi/dfxi/xi) # compute seterusnya xi x1 = xi - fxi/dfxi # print('%i, %15.12f, %15.12f, %15.12f' % (kaunter+1, x1, fxi/dfxi, reldiff)) # output perdagangan untuk input untuk langkah legatan seterusnya xi = x1 # meningkatkan kaunter kaunter += 1 # plot bilangan lelaan yang diperlukan pada x0 plt.plot tertentu (x0,kaunter,'r.', markersize=5) # plt.plot (x0,kaunter,'r.', menandakan=5, label=cetakan print (x0,x1,counter,reldiff) plt.plot (dipilihnx0,counter_chosen,'r.', markersize=5, label=# lelaan) # menambah lagenda ke plot plt.legenda() # menyimpan angka ke fail output plt.savefig ('newton-method-example-tikus-awal-pergantungan.jpg , bbox_inches=tidak ketat') #plt.savefig ('newton-method-example-plot-demo.jpg' , bbox_inches='strict', dpi=300) plt.close() # contact the solution function for func of your choice solve_and_plot_newton(e9) More information about the Newton-RaphsonThere method quite a bit to explain, when it comes to the Newton-Raphson method, or the Newton method. If you like this example, but need more information and in-depth explanation of the step-by-step method, then go to my article Newton Methods Explaining: Details, Pictures, Python Codes and How to Find Early Guess in Newton Methods.If you enjoy cold pictures, complex numbers, and more Newton methods, check out my article Newton Fractals Explains: Examples and Pyons Code.