In [numerical analysis](https://en.wikipedia.org/wiki/Numerical_analysis), **Newton's method**, also known as the **Newton–Raphson method**, named after [Isaac Newton](https://en.wikipedia.org/wiki/Isaac_Newton) and [Joseph Raphson](https://en.wikipedia.org/wiki/Joseph_Raphson), is a [root-finding algorithm](https://en.wikipedia.org/wiki/Root-finding_algorithm) which produces successively better [approximations](https://en.wikipedia.org/wiki/Numerical_analysis) to the [roots](https://en.wikipedia.org/wiki/Root_of_a_function) (or zeroes) of a [real](https://en.wikipedia.org/wiki/Real_number)-valued [function](https://en.wikipedia.org/wiki/Function_(mathematics)).

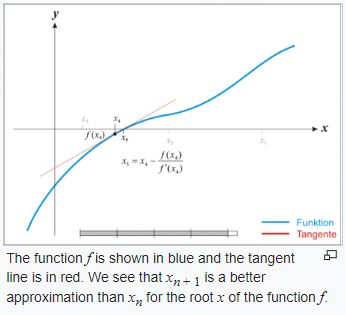
The idea is to start with an initial guess which is reasonably close to the true root, then to approximate the function by its [tangent line](https://en.wikipedia.org/wiki/Tangent_line) using [calculus](https://en.wikipedia.org/wiki/Calculus), and finally to compute the *x*-intercept of this tangent line by [elementary algebra](https://en.wikipedia.org/wiki/Elementary_algebra). This *x*-intercept will typically be a better approximation to the original function's root than the first guess, and the method can be [iterated](https://en.wikipedia.org/wiki/Iterative_method).

More formally, suppose *f* : (*a*, *b*) → ℝ is a [differentiable](https://en.wikipedia.org/wiki/Derivative) function defined on the [interval](https://en.wikipedia.org/wiki/Interval_(mathematics)) (*a*, *b*) with values in the [real numbers](https://en.wikipedia.org/wiki/Real_number) ℝ, and we have some current approximation *xn*. Then we can derive the formula for a better approximation, *xn*+ 1 by referring to the diagram on the right. The equation of the [tangent line](https://en.wikipedia.org/wiki/Tangent_line) to the curve *y* = *f* (*x*) at *x* = *xn* is{\displaystyle y=f'(x\_{n})\,(x-x\_{n})+f(x\_{n}),}



where *f′* denotes the [derivative](https://en.wikipedia.org/wiki/Derivative). The *x*-intercept of this line (the value of *x* which makes *y* = 0) is taken as the next approximation, *xn*+ 1, to the root, so that the equation of the tangent line is satisfied when 





Solving for *xn*+ 1 gives



We start the process with some arbitrary initial value *x*0. (The closer to the zero, the better. But, in the absence of any intuition about where the zero might lie, a "guess and check" method might narrow the possibilities to a reasonably small interval by appealing to the [intermediate value theorem](https://en.wikipedia.org/wiki/Intermediate_value_theorem).) The method will usually converge, provided this initial guess is close enough to the unknown zero, and that *f ′*(*x*0) ≠ 0. Furthermore, for a zero of [multiplicity](https://en.wikipedia.org/wiki/Multiplicity_(mathematics)) 1, the convergence is at least quadratic (see [rate of convergence](https://en.wikipedia.org/wiki/Rate_of_convergence)) in a [neighbourhood](https://en.wikipedia.org/wiki/Neighbourhood_(mathematics)) of the zero, which intuitively means that the number of correct digits roughly doubles in every step. More details can be found in the [analysis section](https://en.wikipedia.org/wiki/Newton%27s_method#Analysis) below.

[Householder's methods](https://en.wikipedia.org/wiki/Householder%27s_method) are similar but have higher order for even faster convergence. However, the extra computations required for each step can slow down the overall performance relative to Newton's method, particularly if *f* or its derivatives are computationally expensive to evaluate.

Program:

import random

import math

def derivat(x):

    a=complex(3\*x\*x-1)

    return a

def poly(x):

    b=complex((x\*x\*x)-x-1)

    return b

def tang(x):

    ans=complex(x-(poly(x)/derivat(x)))

    return ans

print("The polynomial is x^3-x-1")

for i in range(2):

    x=random.randint(-1000,1000)

    y=random.randint(-1000,1000)

    x0=complex(x,y)

    x1=x0

    count=500

    while(count!=0):

        x1=tang(x0)

        x0=x1

        count=count-1

    print()

    print("The root is",x1)

    if(x1.imag!=0):

        x3=complex(x1.real,-x1.imag)

        print()

        print("The root is",x3)

