The Newton-Raphson Method

HILAIRE FERNANDES
hilaire@drgeo.eu

3 June 2019

Abstract

The Newton-Raphson method is an algorithm to search for root of a real function with successive linear approximations. After discussing its historical roots and its mathematics principle, we will propose an implementation in a adaptive environment for interactive geometry and programming. Next, we conclude with its use in the context of secondary education.

1. Introduction

The Newton-Raphson method[3] is a very efficient algorithm to search for the zero of a real function. Most of the time it converges more quickly than a dichotomy approach. Nevertheless, there are some traps to avoid by studying the function and/or its curve.

It is attributed to the mathematicians Newton and Raphson for their respective contributions in 1685 and 1690. However, it is also the result of a slow maturation during the ages, before and after the Newton era. Its definitive principle as known today is from the mathematician Thomas Simpson in 1740. Before, it is derived from the works of mathematicians as the French François Viète in the 16th centuries, Iranians Sharaf al-Din al-Tusi and Jamshīd al-Kāshī in the 13th and 14th centuries. Finally it is a generalization of the Hero of Alexandria method in the 1st century BC also known as the Babylonian method.

2. Hero method

This technique extracts the square root from a positive number *a*. In other words, solving the equation $x^2 = a$ for x or search for the zero of the function $f(x) = x^2 - a$. The method is iterative and geometric. The idea is to search for rectangles getting closer to a square, all of whose surface areas are equal to *a*.

When a first rectangle is chosen with one dimension x, its second dimension is therefore $\frac{a}{x}$, so its surface area is *a* as wished. For the next rectangle, and to get closer to a square, its first dimension is chosen as the average of the two dimensions of the previous rectangle:

$\frac{x+\frac{a}{x}}{2}$. When iterating the process, it gives us the sequence $x_{n+1}=\frac{1}{2}\cdot\left(x_n+\frac{a}{x_n}\right)$ converging to $\sqrt{a}$.

From a programming perspective, it is relatively easy to write a Pharo code in the Dr. Geo[1] workspace:

```
| a x |
a := 2.
x := a / 2.
3 timesRepeat: [x := (x + (a / x)) / 2].
x
```

When executing this program, we get an approximation in the form of a fraction number.

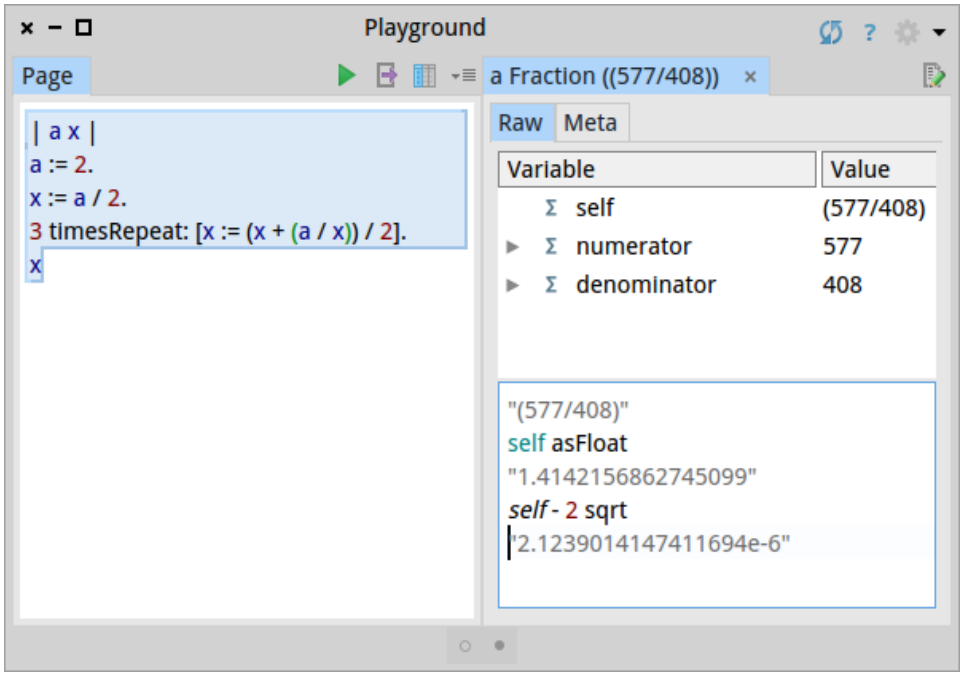


Figure 1. 10^{-5} *approximation at the 3rd iteration.*

It will be fun to visualize this sequence of rectangles converging to a square. A few editing to the previous code gives us the the rectangles. For clarity, we add a supplementary variable *y* to represent the second dimension of the rectangle.

```
| a x y sketch |
a := 2.
x := a / 2.
y := a / x.
sketch := DrGeoSketch new fullscreen axesOn.
(sketch polygon: { 0@0 . x@0 . x@y. 0@y}) name: x.
3 timesRepeat: [
  x := (x + (a / x)) / 2.
  y := a / x.
  (sketch polygon: { 0@0 . x@0 . x@y. 0@y}) name: x asFloat]
```

We can observe the convergence to a square shape is very fast. We have to zoom in to distinguish the last two rectangles. Even with an unfriendly initial value of 0.5 for *x*, the square shape appears quickly.

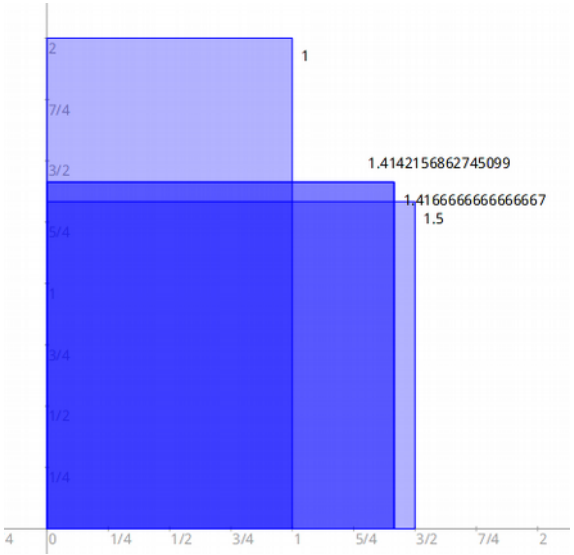


Figure 3. An initial value of 1 for x.

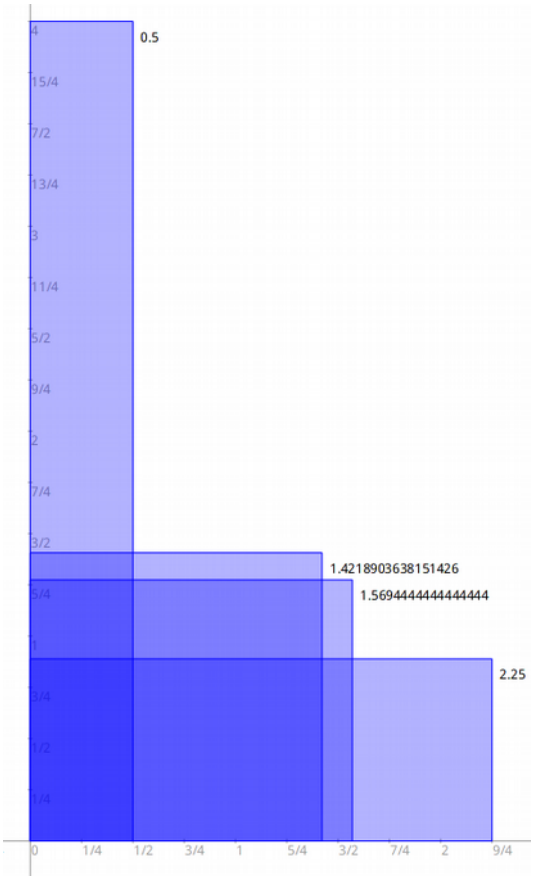


Figure 2. An initial value of 0.5 for x.

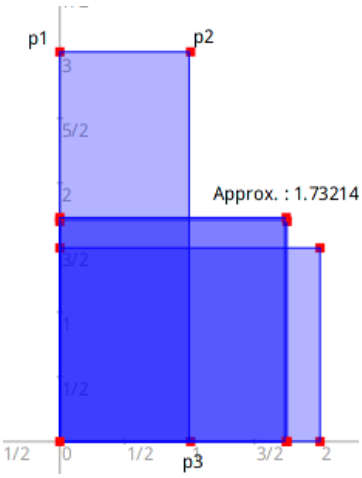
It is more visual but it is not dynamic. In the final sketch, it will be even more fun to modify dynamically the *a* value and the initial *x* value to observe how the convergence works differently according to these different initial parameters. To do so, some more substantial changes are necessary in the program with the use of points – vertexes of the rectangles – whose coordinates are computed by dedicated block of code. This is what makes the resulting sketch dynamic.

```
| a b p1 p2 p3 sketch |
sketch := DrGeoSketch new fullscreen axesOn.
a := sketch integer: 2 at: 0@ -1 from: 2 to: 10
  name: 'a' showValue: true.
b := sketch float: 1 at: 0@ -2 from: 0.1 to: 10
  name: 'Initial value' showValue: true.
p2 := sketch point: [
  | x y |
  x := b value.
  y := a value / x.
  x @ y].
p1 := sketch point: [ :parent | 0 @ parent y] parent: p2.
```

```
p3 := sketch point: [ :parent | parent x @ 0] parent: p2.
sketch polygon: { 0@0 . p1 . p2 . p3}.
3 timesRepeat: [
  p2 := sketch point: [ :parent |
    | x y |
    x := (parent x + (a value / parent x)) / 2.
    y := a value / x.
    x @ y] parent: p2.
  p1 := sketch point: [ :parent | 0 @ parent y] parent: p2.
  p3 := sketch point: [ :parent | parent x @ 0] parent: p2.
  sketch polygon: { 0@0 . p1 . p2 . p3}].
sketch point: [
  p2 name: 'Approx. : ', p2 x asString.
  0@0 ]
```

A few comments are necessary regarding this code:

– The objects `a` and `b` are value sliders. The first one is to select dynamically the *a* value whose square root we want . The second one is to select the initial approximation, the term x_0 of the sequence. To get the value from a value slider we just ask by sending the message `#value`: for example `a value` or `b value`.



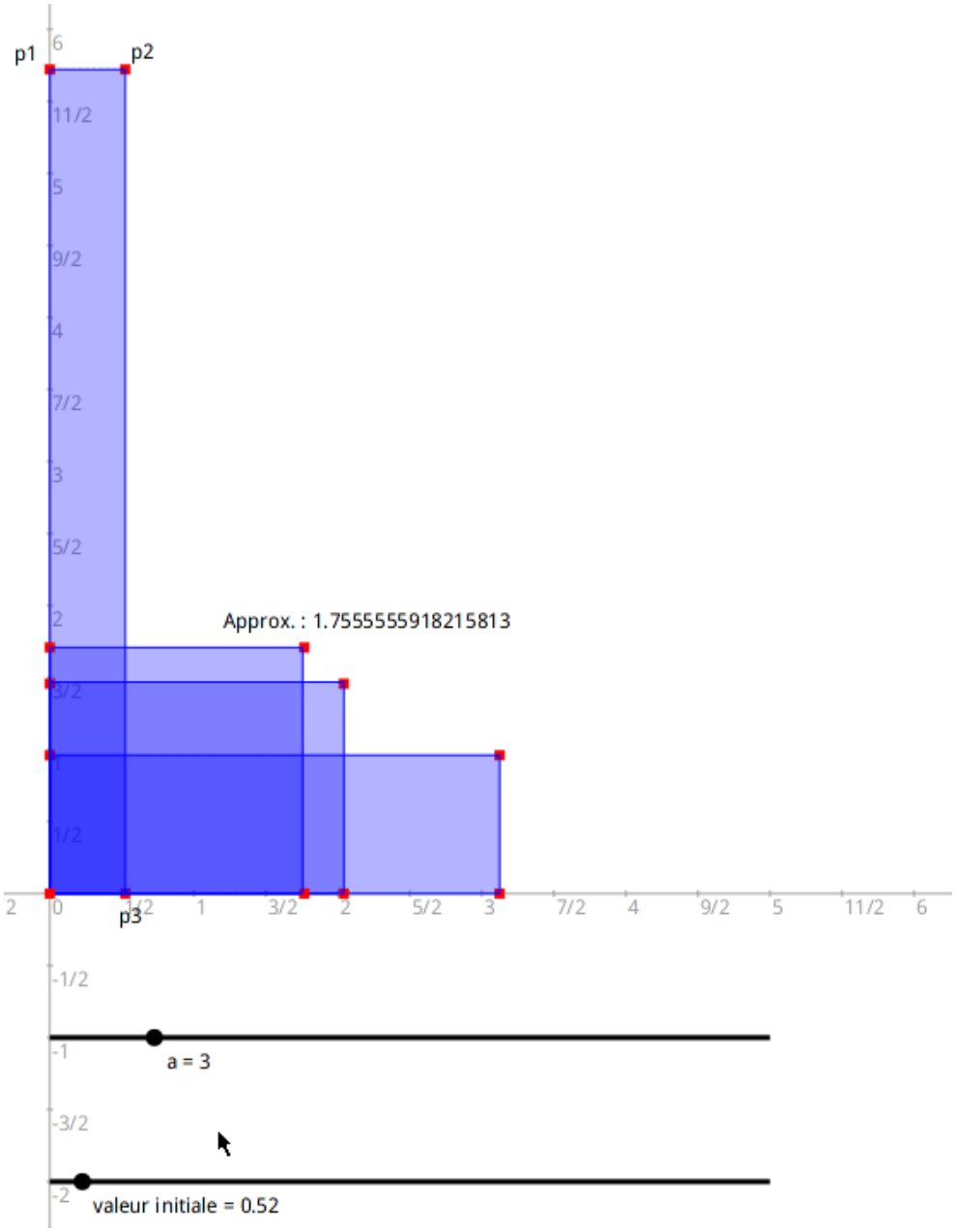
– The vertexes `p2`, `p1` and `p3` are defined one depending on the others. The vertex `p2` is the one holding successively the sequence term x_n and the vertexes `p1` and `p3` are depending on `p2` through coordinate calculus.

– Outside of the loop, the vertex `p2` is defined as a block of code to fetch the values from the value sliders `a` and `b.` There are used to fix the initial term – x, length of the rectangle – and y, high of the rectangle. The vertexes `p1` and `p3` are point defined with `p2` as a parent, therefore the use of the message `#point:parent:` with `p2` as the parent argument.

– In the loop `timesRepeat`, `p2` is defined successively from the previous `p2` object – parent parameter – to fetch the term x_n and to calculate the term x_{n+1} in the sequence.

– Below the loop, the last instruction is a trick to print dynamically the approximation computed after the last term of the sequence is calculated.

The resulting sketch gives us the freedom to interactively modify the integer value *a* whose square root we want to compute. The initial value *x* can be adjusted as well. When playing with this last one, we observe how it is influencing the convergence and how important it is to choose it appropriately.



In the next section, we will examine the Newton-Raphson method and how it is a generalization of the Hero method.

3. Newton-Raphson method

From a geometric point of view this method substitutes a sequence of straight lines to the curve of a function we are searching for a zero. These lines are tangents, the best local linear approximation to the curve. The first tangent line is an important choice. For which x_0 abscissa should it pick up ? How is this choice influencing the convergence to the zero of the function?

In the sketch below, from a first approximation x_0 of the root, the tangent t_1 is constructed. Its intersection with the abscissa axis gives a first linear approximation x_1 of the root. Next it gives us the tangent t_2 to produce a second approximation x_2 and so on. The convergence seems visually very fast: at the 4th iteration it is difficult to distinguish x_4 from the root.

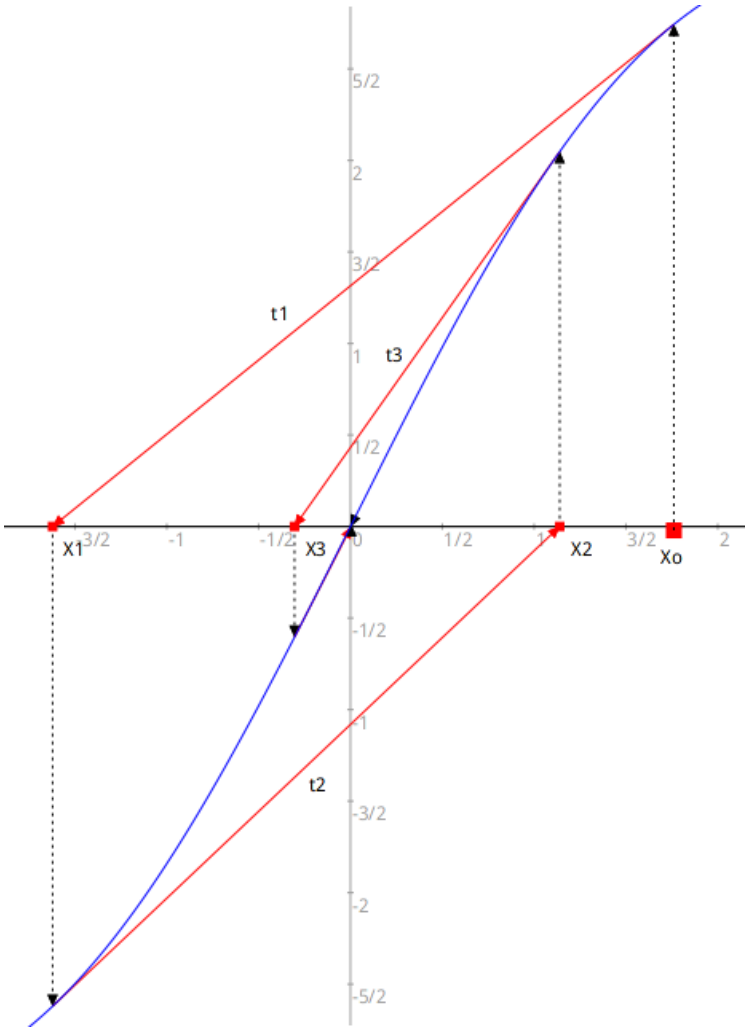


Figure 4. Newton-Raphson method principle

But be careful with the traps, as soon as a tangent is relatively horizontal its intersection with the abscissa axis is pushed far away. When the tangent is horizontal it is catastrophic!

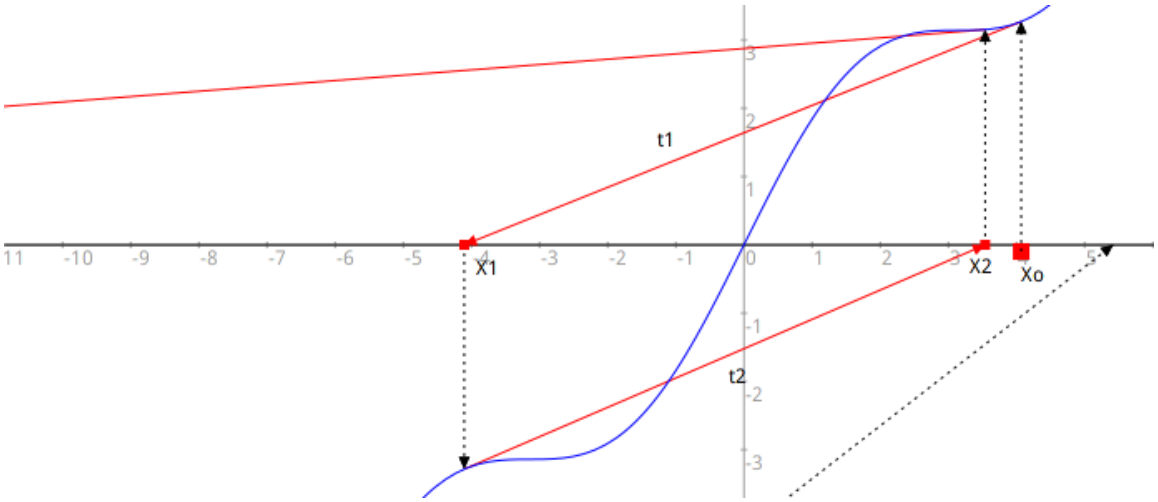


Figure 5. Convergence and divergence are the dilemmas of the method!

In the sketch above, let the the function *f* be plotted in blue. From a first approximation x_0 of the searched root, we deduce the equation of t_1 :

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

The approximation x_1 is deduced as the intersection of t_1 with the abscissa axis:

$$0=f(x_0)+(x_1-x_0)f'(x_0)$$
$$x_1=x_0-\frac{f(x_0)}{f'(x_0)}$$

Then we can generalize with a recurrent sequence:

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

In the case of the Hero method, the *f* function whose zero we are searching for is:

$$f(x)=x^2-a$$

It derivative function is:

$$f'(x)=2x$$

Then the recurrent sequence becomes:

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

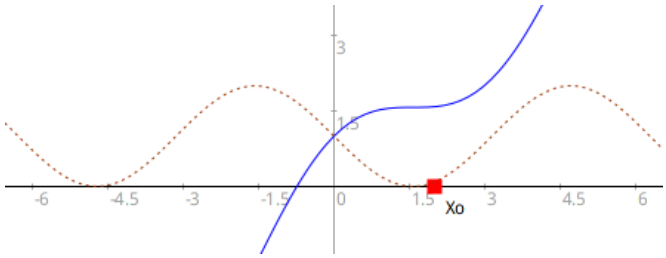
It is the sequence found in the Hero method section at the beginning of the article.

To make the algorithm visual, we need to pick up a function. Let's choose $f(x)=cos(x)+x$, it is funny as it comes with some pitfalls and it is crossing only once the abscissa axis. Of course we want to plot it.

We also need its derivative function, it is easy to calculate it algebraically but we are using the derivative of the poor with numeric approximation. Nevertheless it proves to be useful and it requires no additional effort when we decide to change *f* 's definition.

```
| sketch f df X0 Xn Mn|
sketch := DrGeoSketch new fullscreen axesOn.
f := [:x | x cos + x ].
df := [:x | (f value: x + 1e-8) - (f value: x) * 1e8].
sketch plot: f from: -20 to: 20.
(sketch plot: df from: -20 to: 20) dotted color: Color brown.
X0 := 2.
Xn := (sketch point: X0@0) large; name: 'Xo'.
```

Little unnecessary refinement, we also plot the derivative function.



A few comment regarding this code :

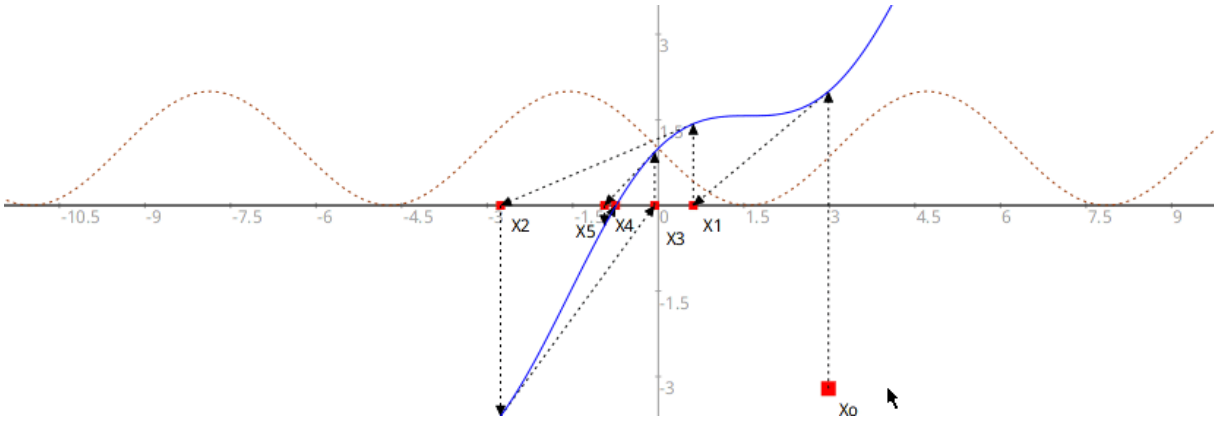
- A function is represented as a block of code with one argument, here `x`.
- The code `x cos` should be understood as sending the message `#cos` to the object `x` to ask its cosine value.
- To understand the definition of the derivative function in the `df` block of code, one need to understand the priority with Pharo messages, see the resource in the annex[2].
- The point `Xn` holds in its abscissa the terms of the sequence x_n , the successive approximations of the root.
- The point `Mn` – not used in the previous code – represents the different tangent points to the curve.

To construct the sequence in the code, we add a loop where the vertexes `Xn` and `Mn` are computed:

```
1 to: 5 do: [ :n |
  Mn := sketch
    point: [ :point | point x @ (f value: point x)]
    parent: Xn.
  Xn := sketch
    point: [ :point |
      point x - ( (f value: point x) / (df value: point x) ) @ 0]
    parent: Mn.
  Xn name: 'X', n asString]
```

To explore interactively the behavior of the method, the points are defined with blocks of code to compute their coordinates. These blocks come with one parameter, the previous term

in the recurrent sequence. To trace the algorithm flow, we construct segments with arrows linking the points `Xn` and `Mn`. Some situations make the algorithm diverge.



4. Conclusion

In our study of the Newton-Raphson method, we first get interested by the Hero method in the calculus of the square root. It was easily transposed in a 5 lines source code, understandable by secondary I & II students. Next, to illustrate the geometric nature of this algorithm, we extend the code with the construction of the successive rectangles converging toward the solution square. Then we deduce the root as the side length of this square. Again the modifications to the code are in the scope of the students. Finally to explore dynamically the convergence of the algorithm according to its initial parameters, the code is extended for a second time with the introduction of blocks of code to define points. This addition is a small conceptual leap, but it is not weighed down by the difficulty of the recurrence as it was already introduced in the previous version of the code.

The Newton-Raphson method generalizes the Hero method. It was programmed with geometric and analytic approaches. Again the conception of a dynamic sketch to explore the algorithm behavior is possible thanks to the use of geometric point defined with block of code.

In the context of programming teaching, this approach starting with the Hero method and concluding with the Newton-Raphson method introduces progressively several concepts in programming while doing mathematics at the same time: variable, loop, lambda function – block of code, use of the abstraction of object oriented programming.

In the context of mathematics teaching, we show how a teacher can produce in a few line of codes in a dynamic document to illustrate mathematics knowledge.

The use of the block of code, also named lambda function, gives this creative capability to enrich applications and the produced sketches with dynamic behaviors. Dan Ingalls the principal architect of the Smalltalk language defines it as: "*The purpose of the Smalltalk project is to provide computer support for the creative spirit in everyone.*"[4]

As a matter of comparison, the table below shows the lambda function in some languages.

Pharo	Python	Javascript
f := [:x | x cos + x]	f = lambda x: math.cos(x) + x	const f = x => Math.cos(x) + x;
df := [:x | (f value: x + 1e-8) - (f value: x) * 1e8]	df = lambda x: (f(x + 1e-8) - f(x)) * 1e8	const df = x => (f(x + 1e-8) - f(x)) * 1e8;

ANNEXE A

The complete program for the interactive sketch of the Newton-Raphson method.

```
| sketch f df X0 Xn Mn|
sketch := DrGeoSketch new fullscreen axesOn.
f := [ :x | x cos + x ].
df := [ :x | (f value: x + 1e-8) - (f value: x) * 1e8].
sketch plot: f from: -20 to: 20.
(sketch plot: df from: -20 to: 20) dotted color: Color brown.
X0 := 2.
Xn :=  (sketch point: X0@0) large; name: 'Xo'.
1 to: 5 do: [ :n |
  Mn := sketch
    point: [ :point | point x @ (f value: point x)]
    parent: Xn.
  Mn hide.
  (sketch segment: Xn to: Mn) dotted backArrow .
  Xn := sketch
    point: [ :point |
      point x - ( (f value: point x) / (df value: point x) ) @ 0]
    parent: Mn.
  ( sketch segment: Mn to: Xn) dotted backArrow.
  Xn name: 'X', n asString]
```

Resources

[1] Dr. Geo, interactive geometry and programming environment

[2] Syntaxe Pharo (see messages)

[3] The Newton-Raphson Mehod

[4] Design Principles Behind Smalltalk