Zeros of functions

- Problem:

  suppose you want to find the value of x which satisfies the relation:

$$f(x) = 0 \qquad x \in [a, b]$$

  namely the **root** of the equation!

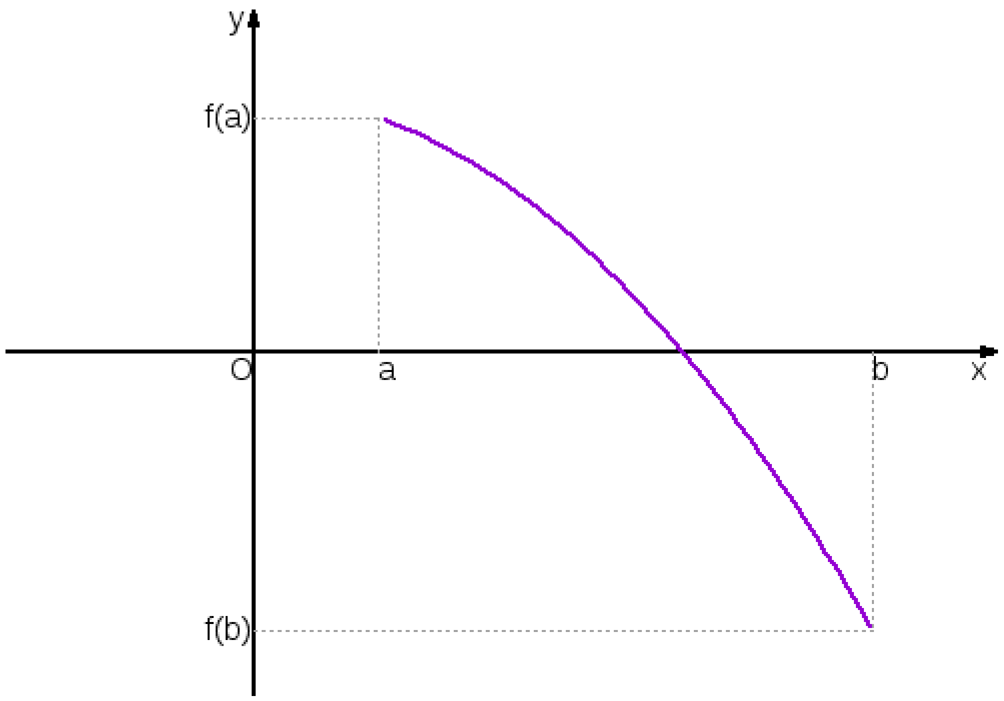
- **Bolzano's theorem** (or intermediate value theorem):

  If $f(x)$ is a continuous function inside $[a,b]$ and it takes values $f(a)$ and $f(b)$ in a and b, then it also takes any value between $f(a)$ and $f(b)$ at some point inside the interval!

Zeros of functions

- An important corollary of this theorem is that:

  if the function changes its sign in $f(a)$ and $f(b)$, then it **must** have a **root** inside $[a,b]$!



Zeros of functions

- A very immediate method to find the root of the equation could then be:
  - Fix the precision ε with which one wants to find the solution;
  - Divide $[a,b]$ in N subsequent subintervals $[x_i,x_{i+1}]$, $i=0, \ldots, N$ and $x_0=a$, $x_N=b$, $\Delta x = |x_{i+1} - x_i| \leq \epsilon$;
  - Scan all the subintervals and find the one for which:

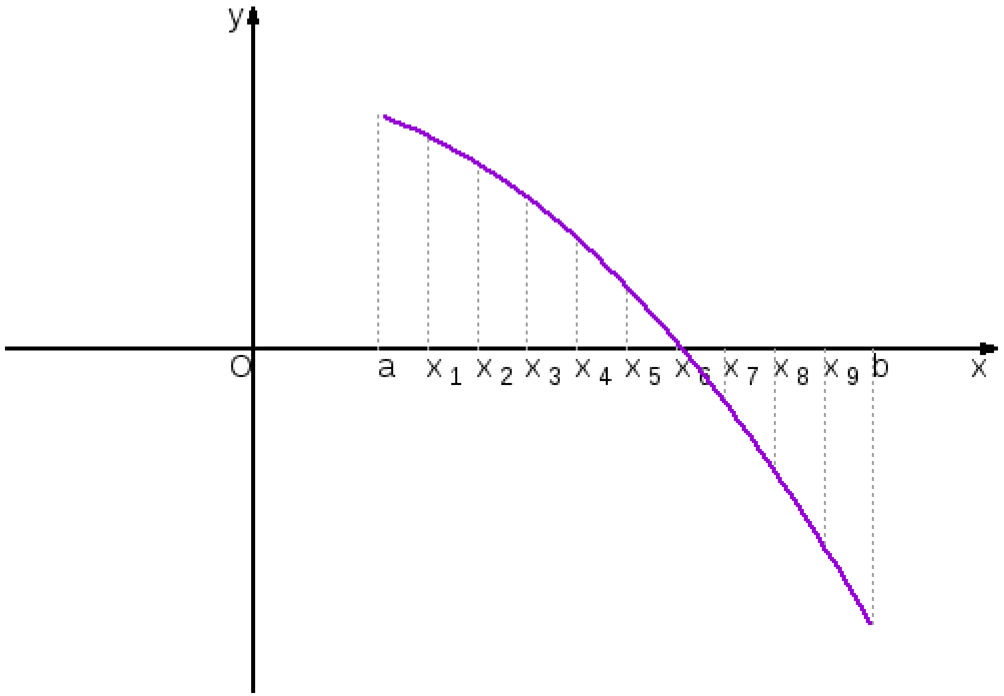
$$f(x_i) \cdot f(x_{i+1}) < 0$$

  - Now we have determined the zero with a precision equal to Δx:

$$\alpha = \frac{1}{2}(x_i + x_{i+1}) \pm \frac{1}{2}\Delta x$$

Zeros of functions

- In a graphical representation for N=10:



- The solution is: $\alpha = (x_6+x_7)/2$

Zeros of functions

- This is a good method only when the number *N* of subintervals is not too high, since for each subinterval $[x_i, x_{i+1}]$ one has to evaluate the product $f(x_i) \cdot f(x_{i+1})$ *N* times, in the worst case!
- A more efficient algorithm: **the bisection algorithm**!
  - It belongs to the class of "**divide-and-conquer**" algorithms
  - Like other algorithms of the same kind (e.g. FFT, we will see later) lowers the number of required operations.

Bisection algorithm

- The idea is to build up a sequence of smaller and smaller subintervals by halving the interval at the previous step!
- At the first step of the sequence, one starts with an interval a_0=a, b_0=b.
- Then one evaluates the midpoint x_0 between a_0 and b_0: $$x_0 = \frac{1}{2}(a_0 + b_0)$$
- Now, unless the zero is exactly in x_0, it stays either in [a_0,x_0] or in [x_0,b_0].

Bisection algorithm

- This is realized by looking at the products:

$$f(a_0) \cdot f(x_0) \qquad \text{and} \qquad f(x_0) \cdot f(b_0)$$

- One of the two **has to be** < 0!
- In case it is the first one, we let:

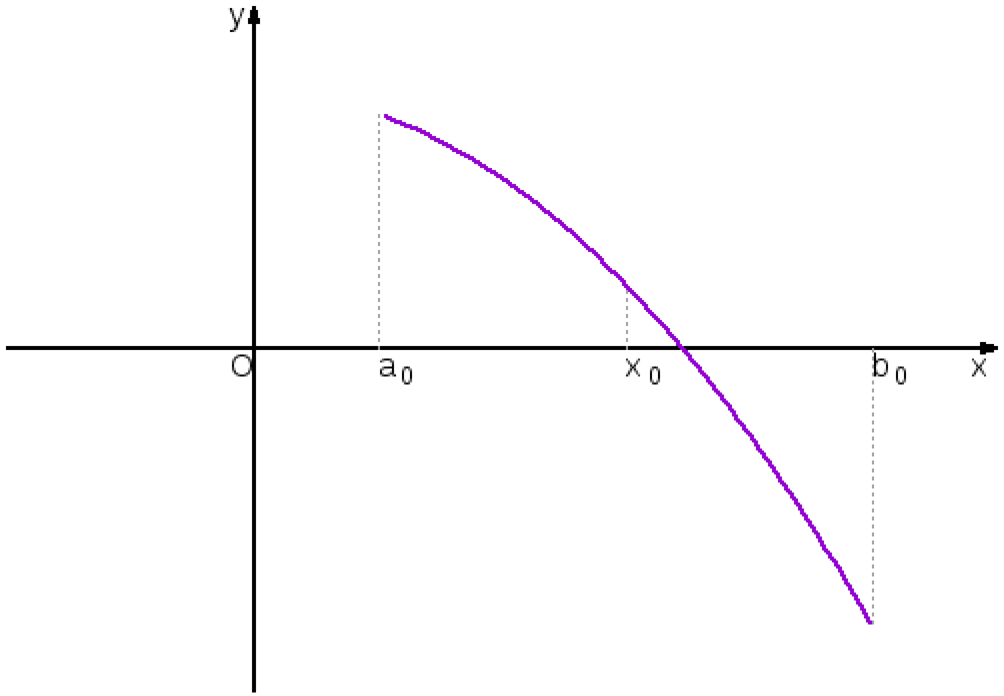
$$a_1 = a_0,\ b_1 = x_0;$$

- In case it is the second one, we let:

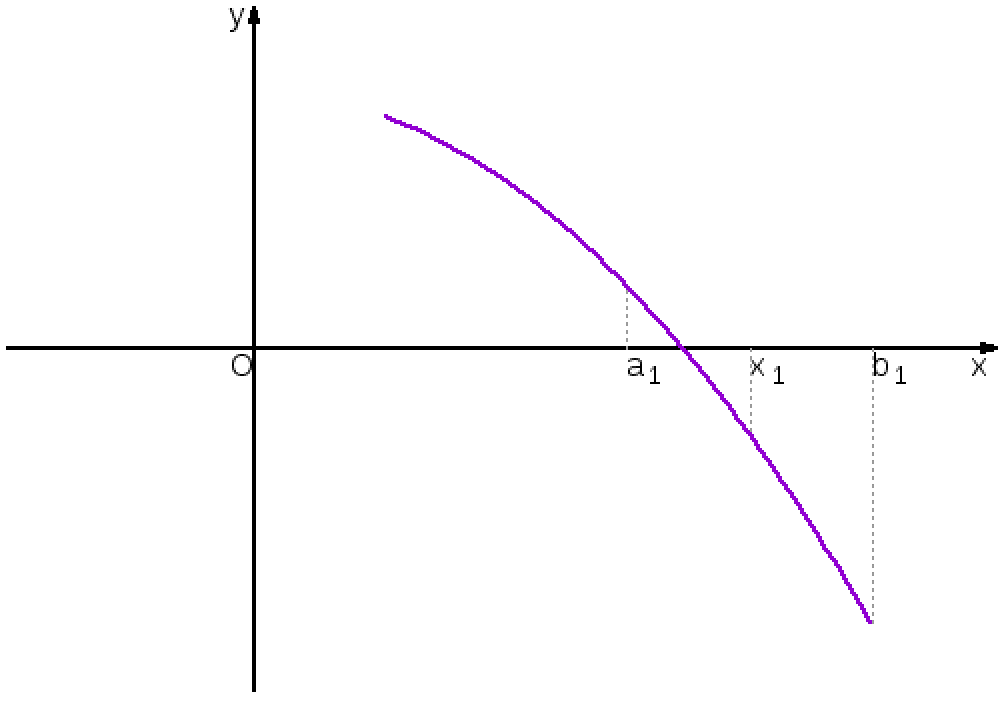
$$a_1 = x_0,\ b_1 = b_0;$$

and so on…

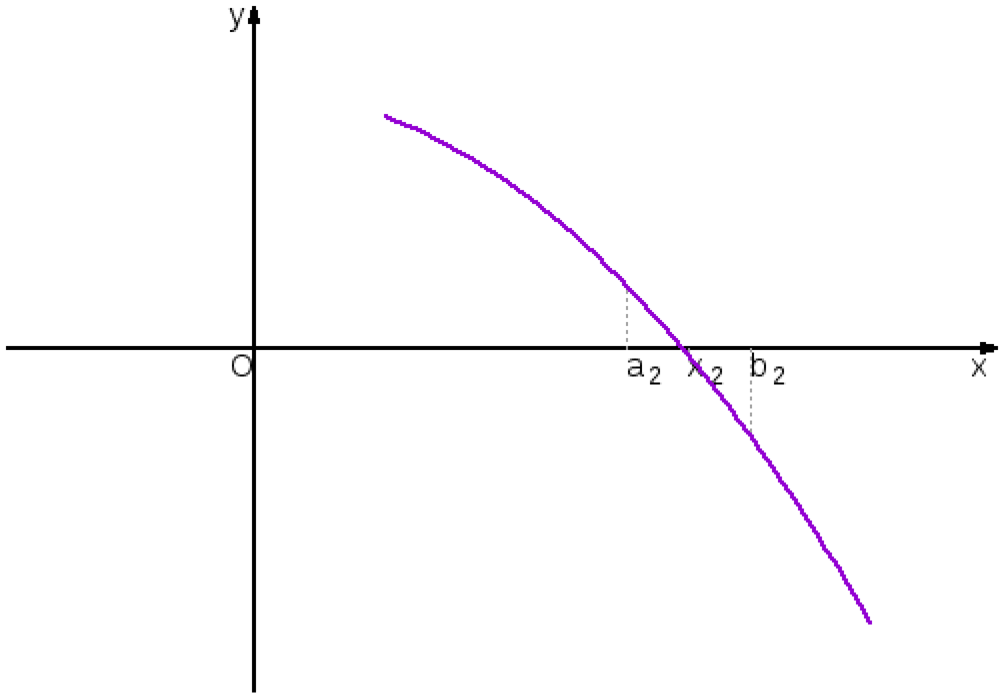
Bisection algorithm



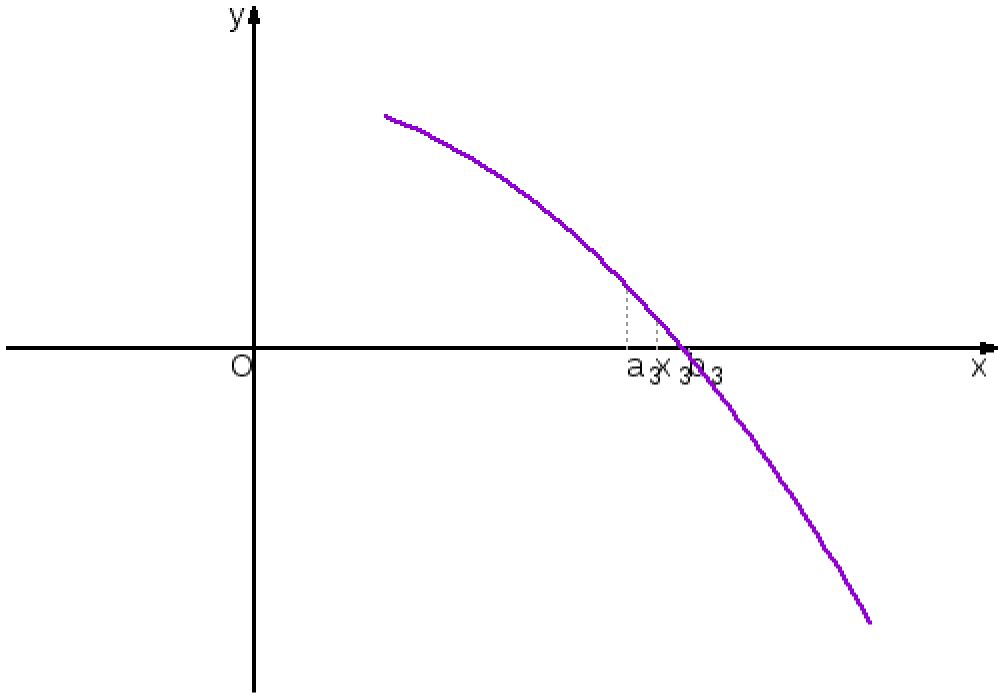
Bisection algorithm



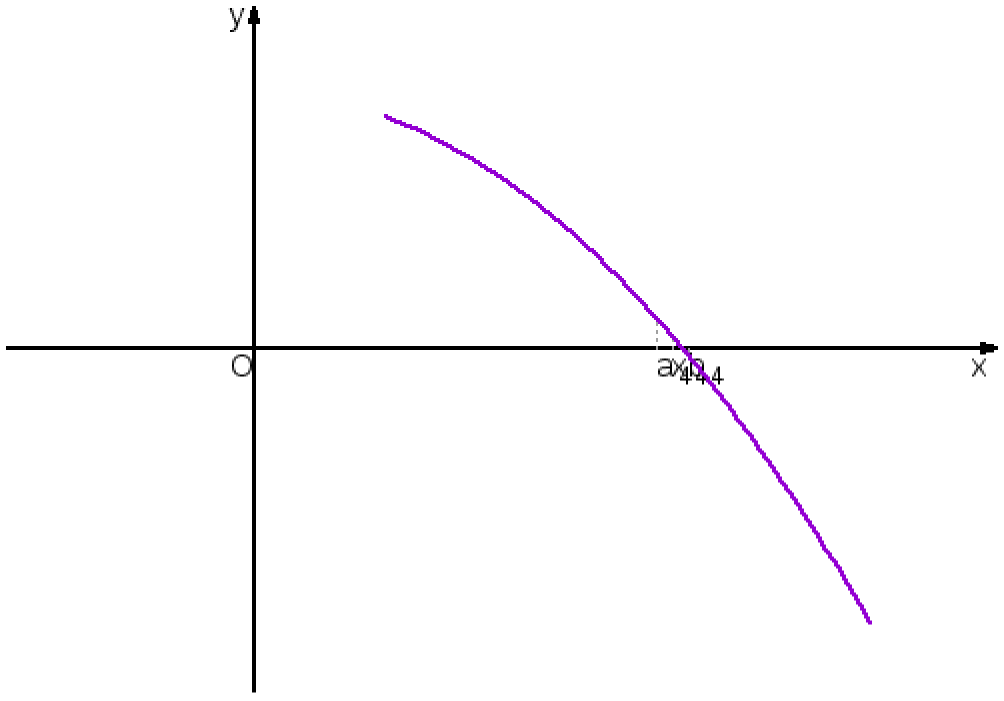
Bisection algorithm



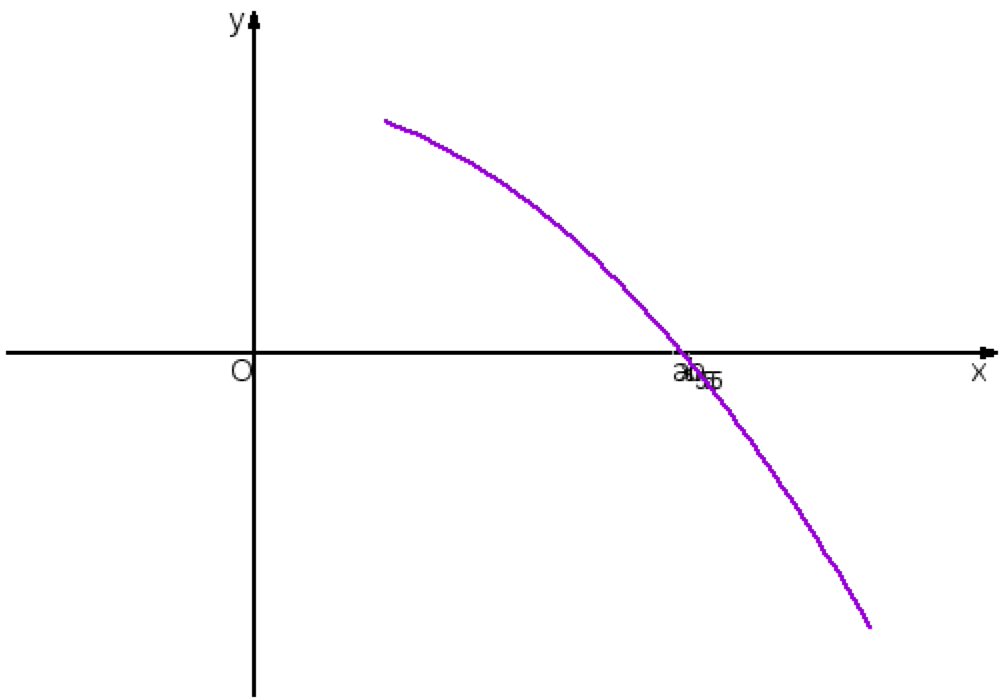
Bisection algorithm



Bisection algorithm



Bisection algorithm



Bisection algorithm

- How many iterations are needed in order to compute the value of the zero with a given precision ε?
- We know that, at each iteration, the interval is halved, namely the width w_i of the interval in which the zero is at the i-th iteration is:

$$w_0 = b_0 - a_0 = b - a \qquad \text{at iteration} \quad i = 0;$$

$$w_1 = b_1 - a_1 = \frac{1}{2} w_0 = \frac{1}{2}(b - a) \qquad \text{at iteration} \quad i = 1;$$

$$w_2 = b_2 - a_2 = \frac{1}{2} w_1 = \frac{1}{2^2}(b - a) \qquad \text{at iteration} \quad i = 2;$$

$$\ldots$$

Bisection algorithm

- At the n-th iteration we have:

$$w_n = \frac{1}{2^n}(b-a)$$

- We stop the procedure when:

$$\begin{aligned}
w_n \le \epsilon \quad &\Rightarrow \quad \frac{1}{2^n}(b-a) \le \epsilon \\
&\Rightarrow \quad \log_2\left[\frac{1}{2^n}(b-a)\right] \le \log_2 \epsilon \\
&\Rightarrow \quad -\log_2(2^n) + \log_2(b-a) \le \log_2 \epsilon \\
&\Rightarrow \quad \log_2\left[\frac{(b-a)}{\epsilon}\right] \le n
\end{aligned}$$

Bisection algorithm

- Pros:
  - The algorithm **always gives a solution** (provided that the initial interval does indeed contains a zero!)!
  - The **number of iterations** required **is known** a priori independently of the particular function *f(x)*!
- Cons:
  - The algorithm can be **very slow** (large n!) if the initial interval (b-a) is large and the required precision (ε) is small!

Newton's algorithm

- The idea of Newton's algorithm is to find a succession of subsequent approximations of the solution.
- Lagrange's mean value theorem:

  Let be $f(x)$ a real function which is continuous and differentiable on the interval $[a,b]$. Then, there exists some c in $[a,b]$ such that:

$$f'(c) = \frac{f(b) - f(a)}{b - a}$$

Newton's algorithm

- Now, let α be the root of the equation and x_0 a value close to α. Then, by applying the previous theorem to the interval $[x_0,\alpha]$:

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

  that is, there must be a value c in $[x_0,\alpha]$ which satisfies the equation:

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

  since $f(\alpha) = 0$!

Newton's algorithm

- Of course, if we knew the point c, we would have solved the problem, since the relation above gives us the value of α.
- We can then try to find a new value for the solution, x_1, by approximating c with x_0:

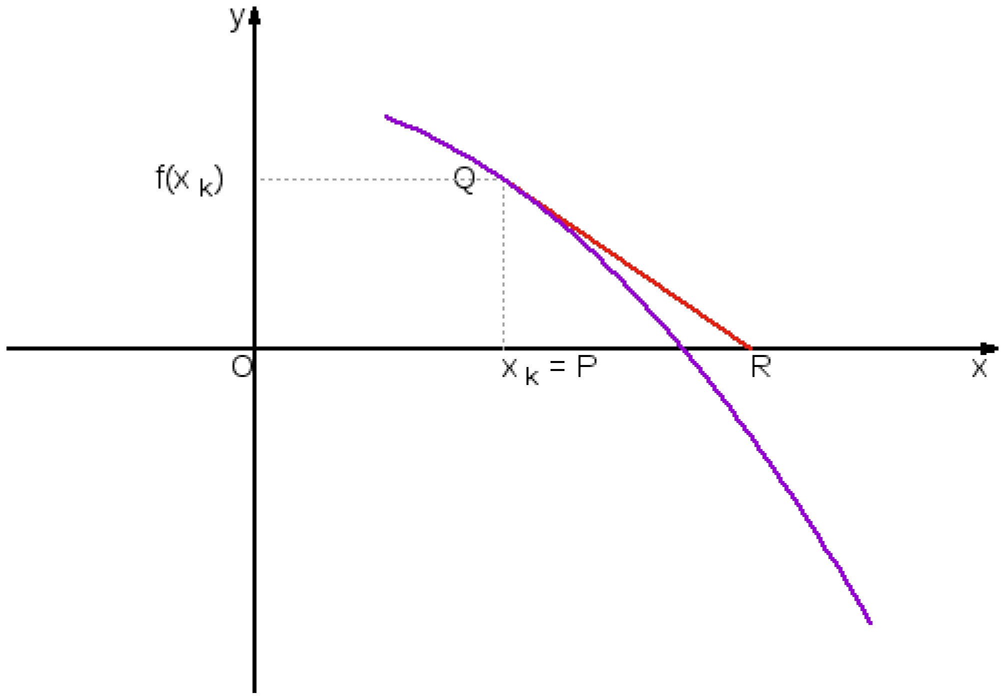
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

- In an analogous way, we can build up a successions of values (hopefully!) closer and closer to the solution, as:

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

Newton's algorithm

- Geometrical interpretation:



$$f(x_k) = |PQ|$$

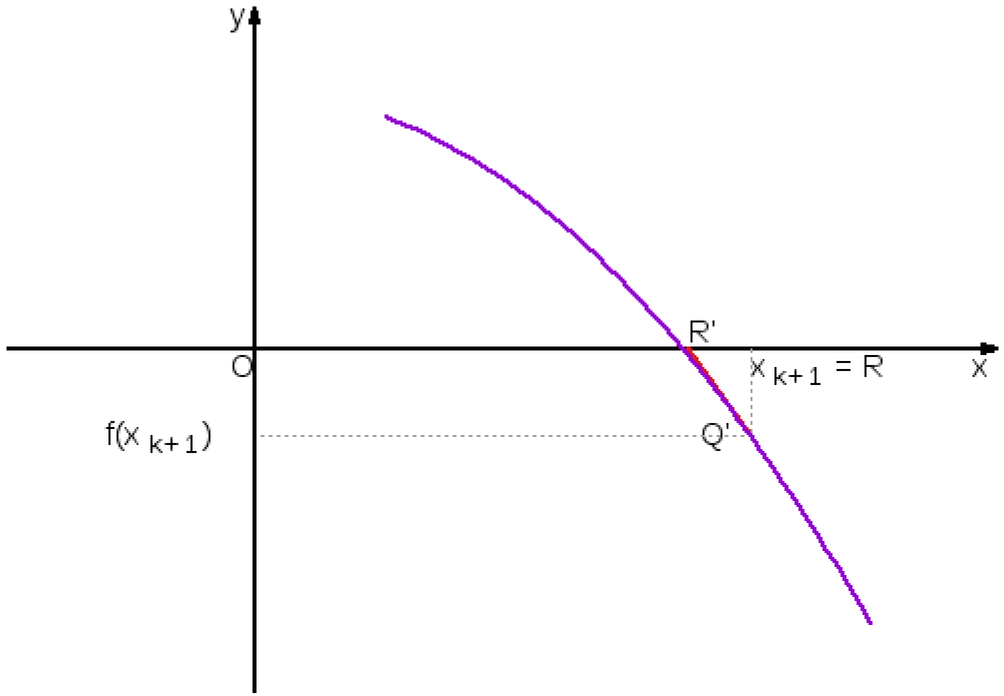
$$f'(x_k) = -\frac{|PQ|}{|RP|}$$

$$x_k = |OP|$$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = |OP| - \frac{|PQ|}{-|PQ|/|RP|} = |OP| + |RP| = |OR|$$

Newton's algorithm

- At the subsequent step:



$$f(x_{k+1}) = |RQ'|$$

$$f'(x_{k+1}) = \frac{|RQ'|}{|RR'|}$$

$$x_{k+1} = |OR|$$

$$x_{k+2} = x_{k+1} - \frac{f(x_{k+1})}{f'(x_{k+1})} = |OR| - \frac{|RQ'|}{|RQ'|/|RR'|} = |OR| - |RR'| = |OR'|$$

Newton's algorithm

- Pros:
  - The convergence of the method is **very** fast!
  - It does not depend on the width of the interval in which the root is located, but only on the **initial guess** x_0;
- Cons:
  - However, the method may also not converge, it depends on the **shape of the function** $f(x)$;
  - The number of iterations is not fixed a priori, but depends on x_0!

Newton's algorithm

- Two examples:
  - Find the root of the equation:

$$e^x - 1.5 = 0$$

    in the interval [0,3];

  - Find the root of the equation:

$$\frac{x}{3/2 + \sin(\pi x)} - \frac{1}{2} = 0$$

    in the interval [0,6].

Newton's algorithm

- In the first case:

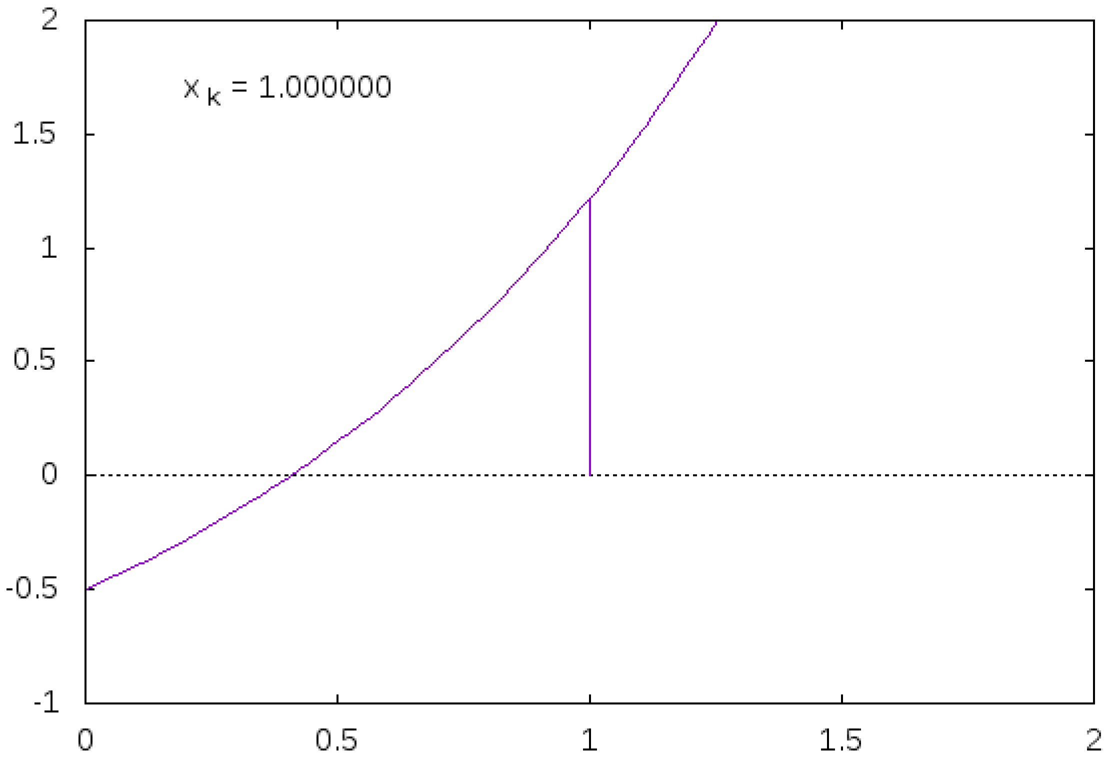
$$f(x) = e^x - 1.5;$$
$$f'(x) = e^x.$$

- In the second one:

$$f(x) = \frac{x}{3/2 + \sin(\pi x)} - \frac{1}{2};$$
$$f'(x) = \frac{3/2 + \sin(\pi x) - x\pi\cos(\pi x)}{[3/2 + \sin(\pi x)]^2}$$

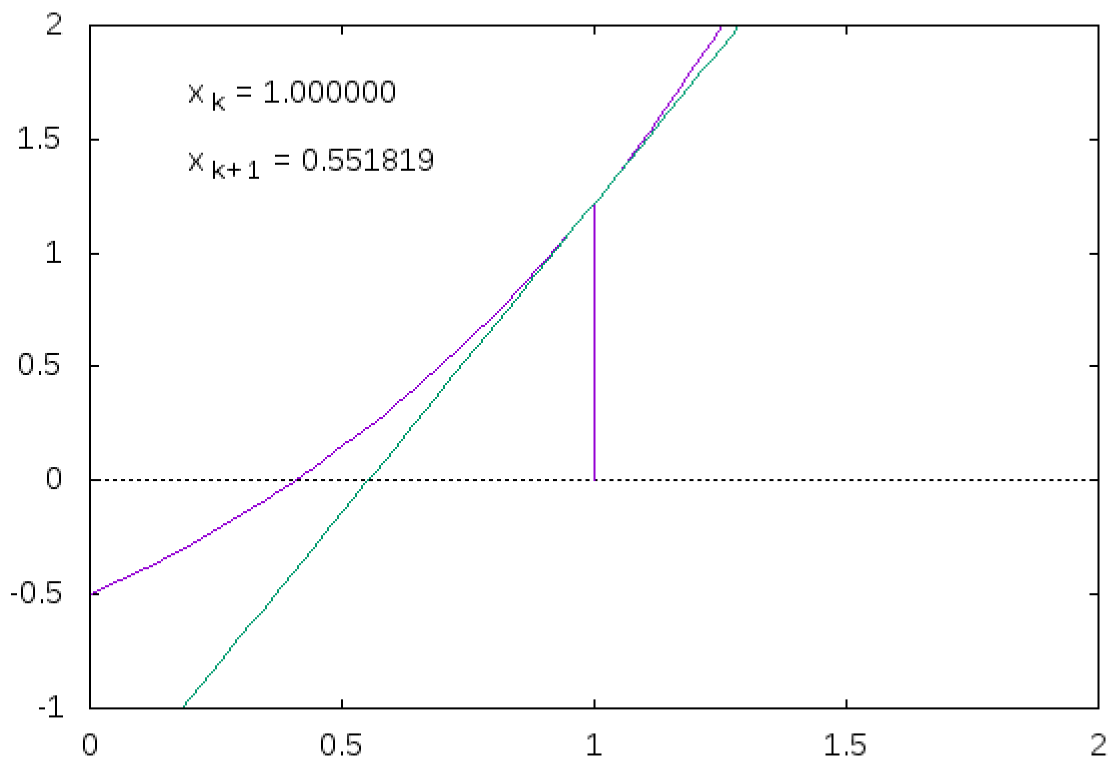
Newton's algorithm

- First case: x_0=1, ε =10^{-4}



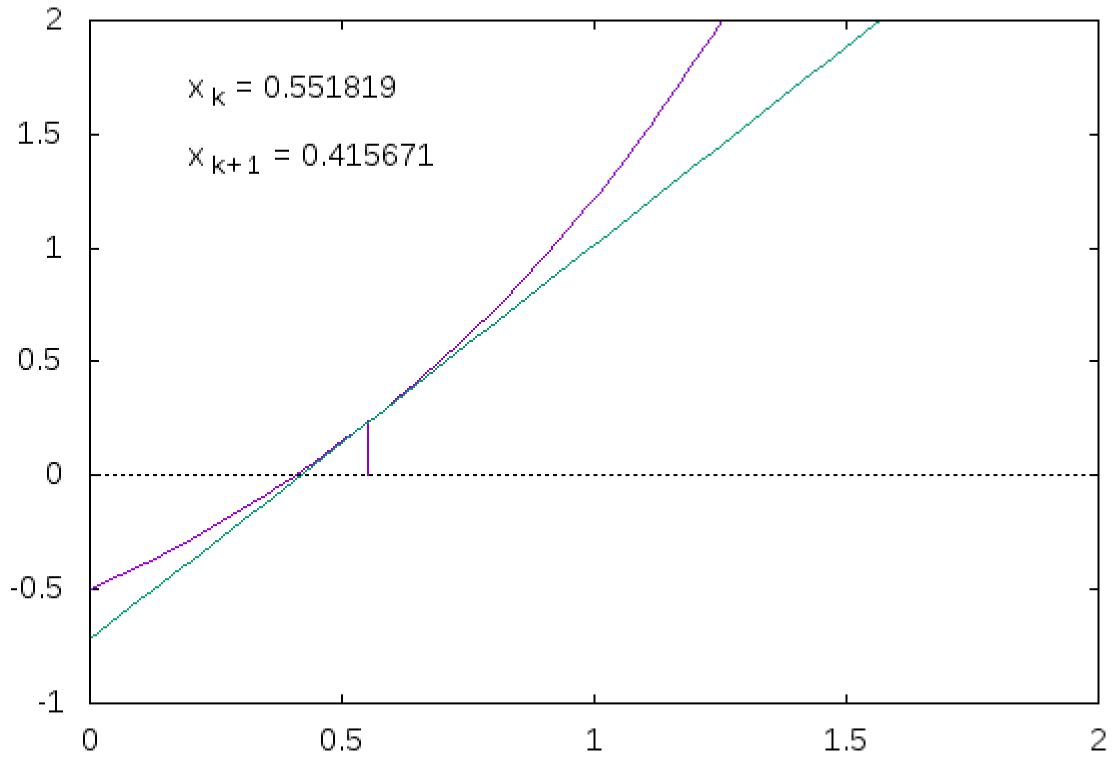
Newton's algorithm

- First case: x_0=1, ε =10^{-4}



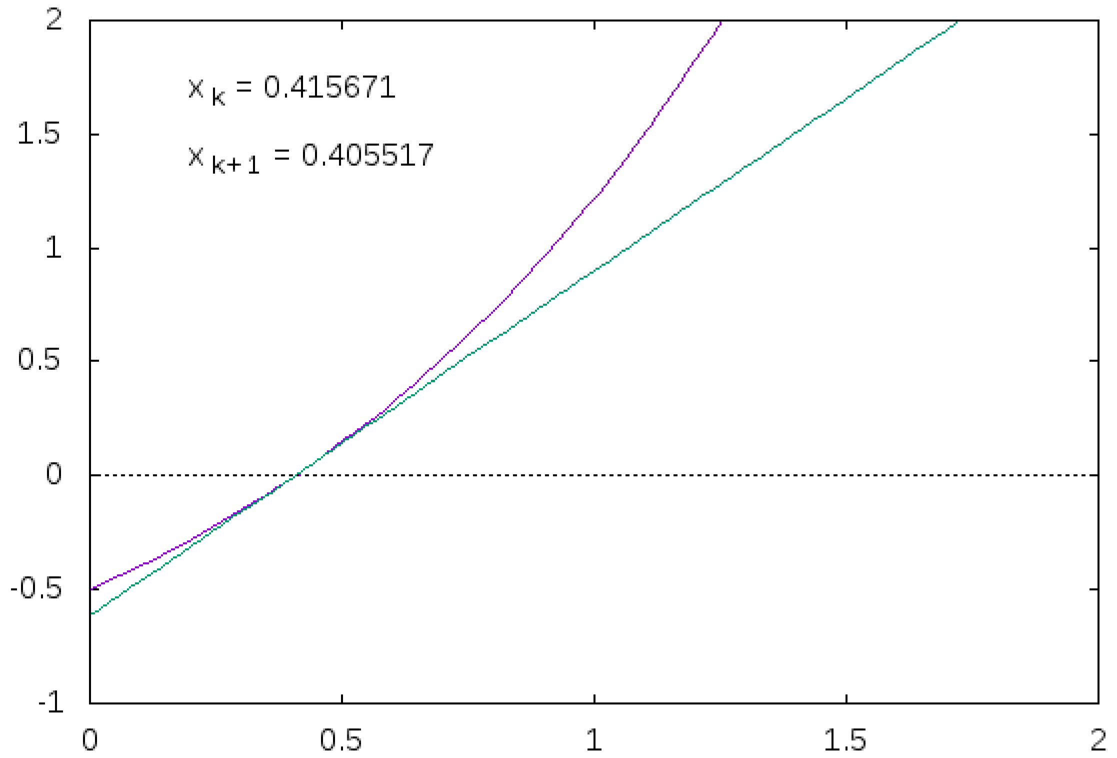
Newton's algorithm

- First case: x_0=1, ε =10^{-4}



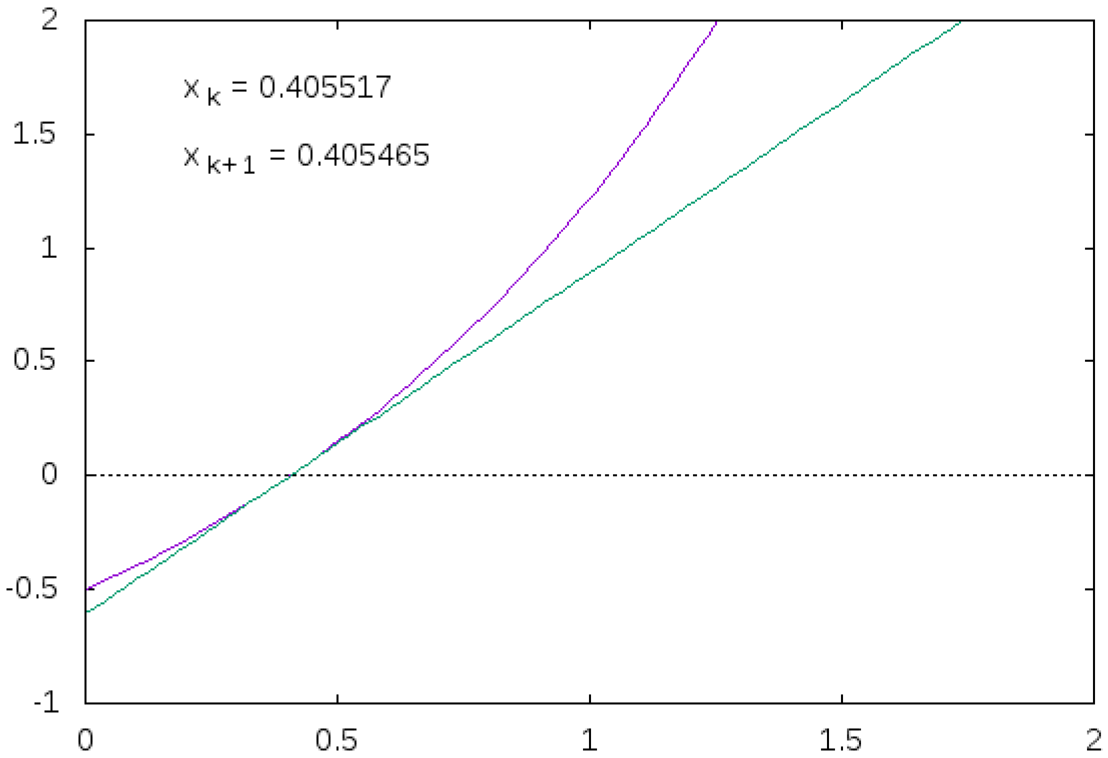
Newton's algorithm

- First case: x_0=1, ε =10^{-4}



Newton's algorithm

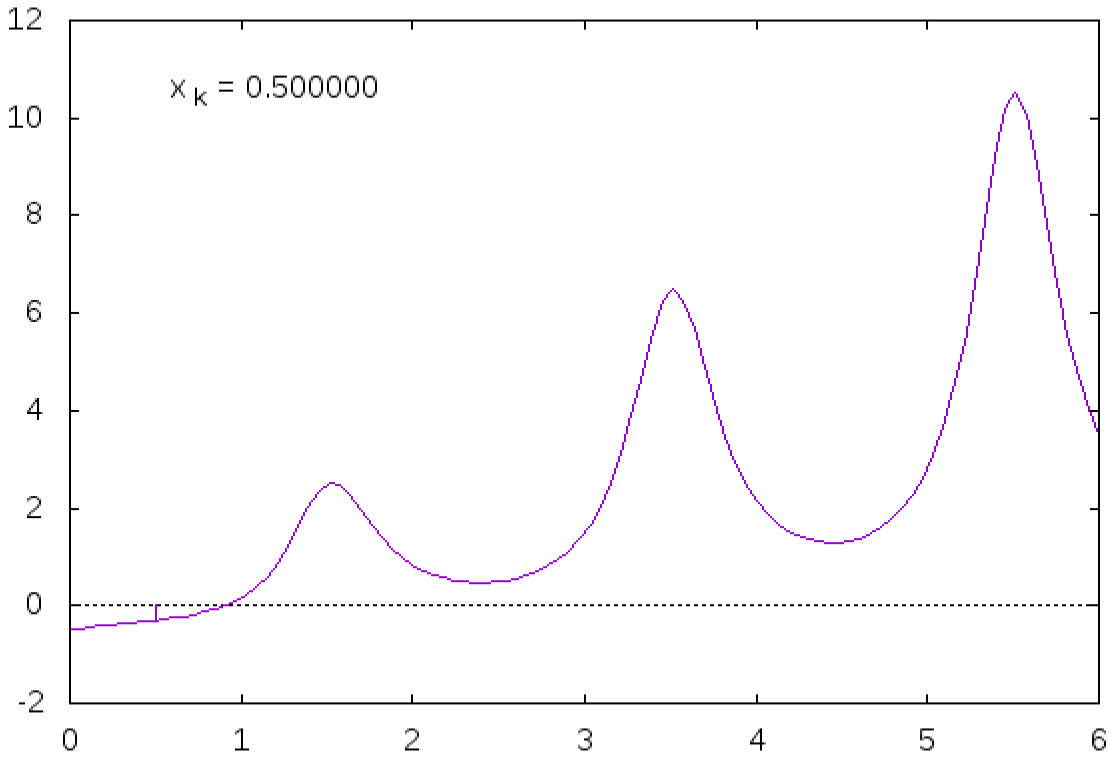
- First case: x_0=1, ε=10^{-4}



$$\alpha = \log(3/2) = 0.405465108108164438197$$

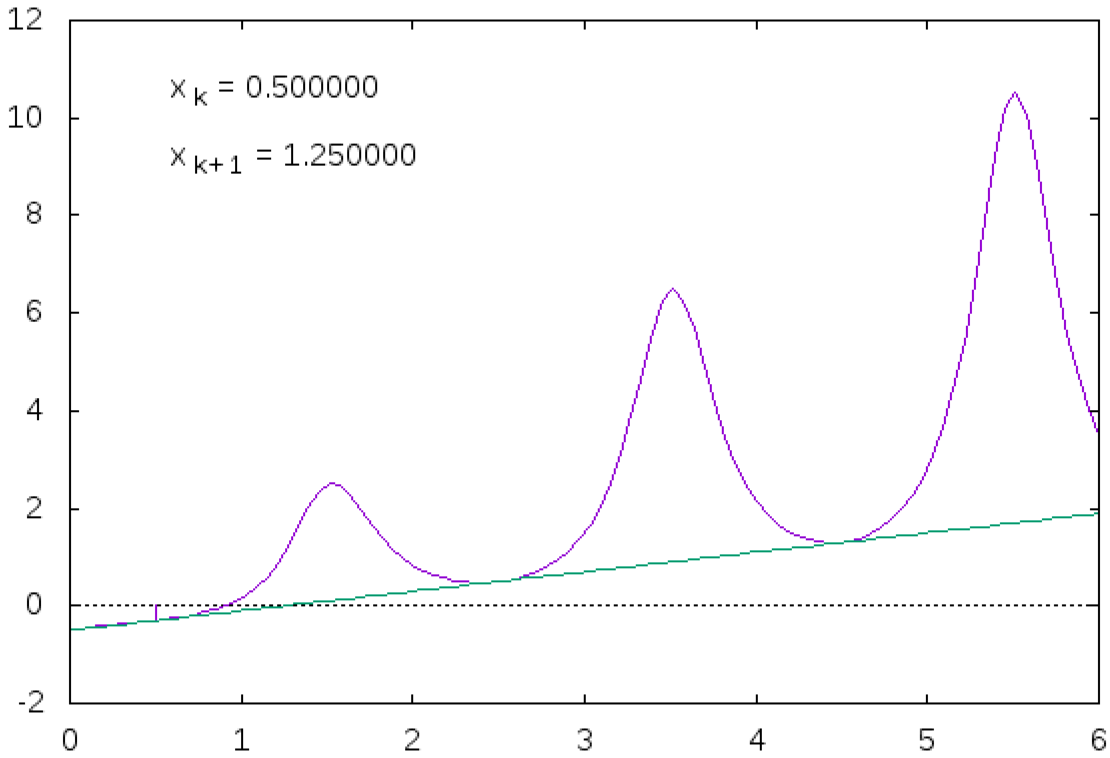
Newton's algorithm

- Second case: x_0=0.5, ε =10^{-4}



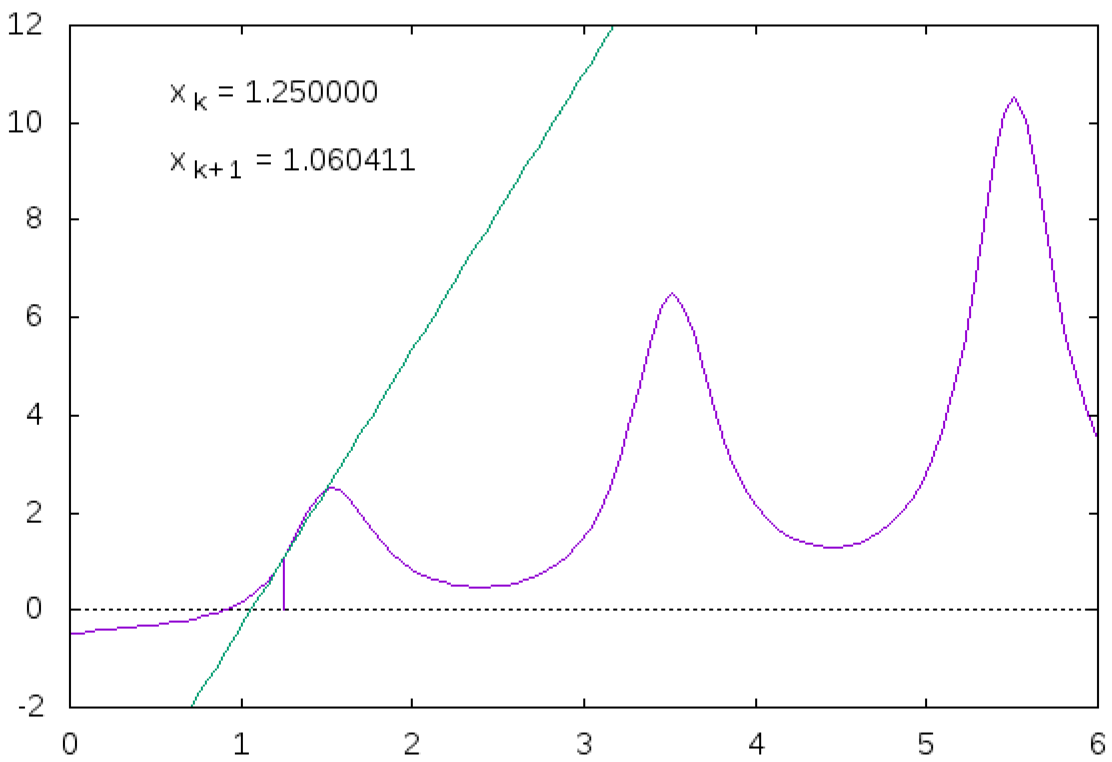
Newton's algorithm

- Second case: x_0=0.5, ε =10^{-4}



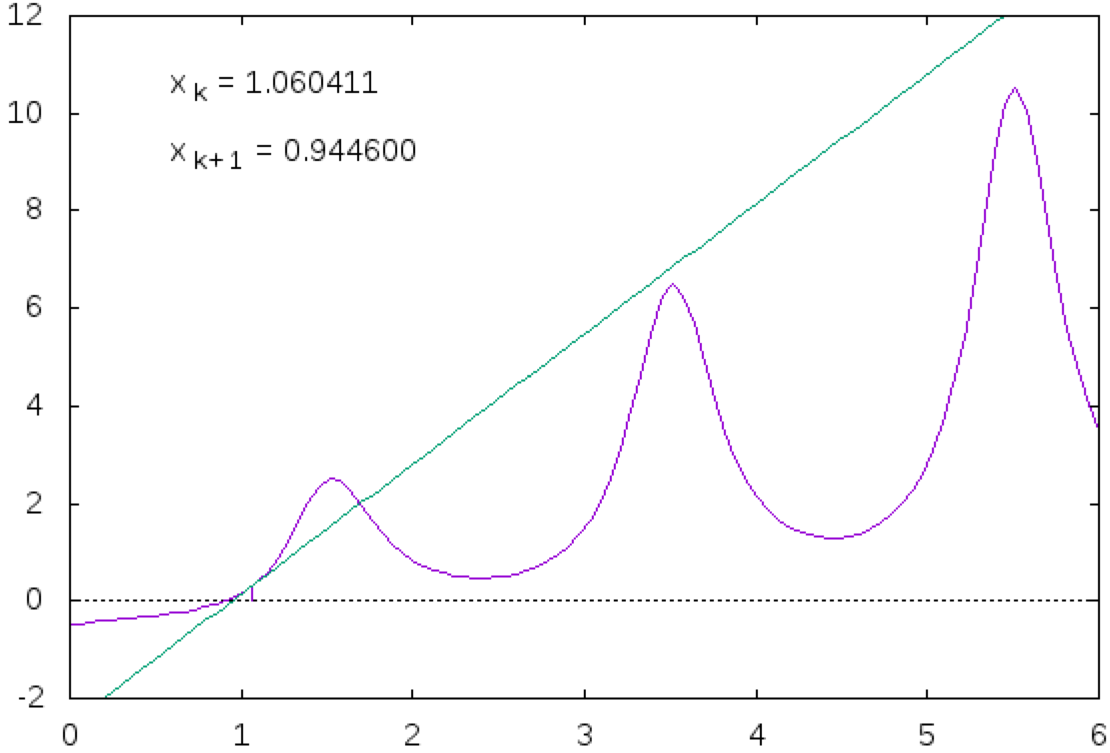
Newton's algorithm

- Second case: x_0=0.5, ε =10^{-4}



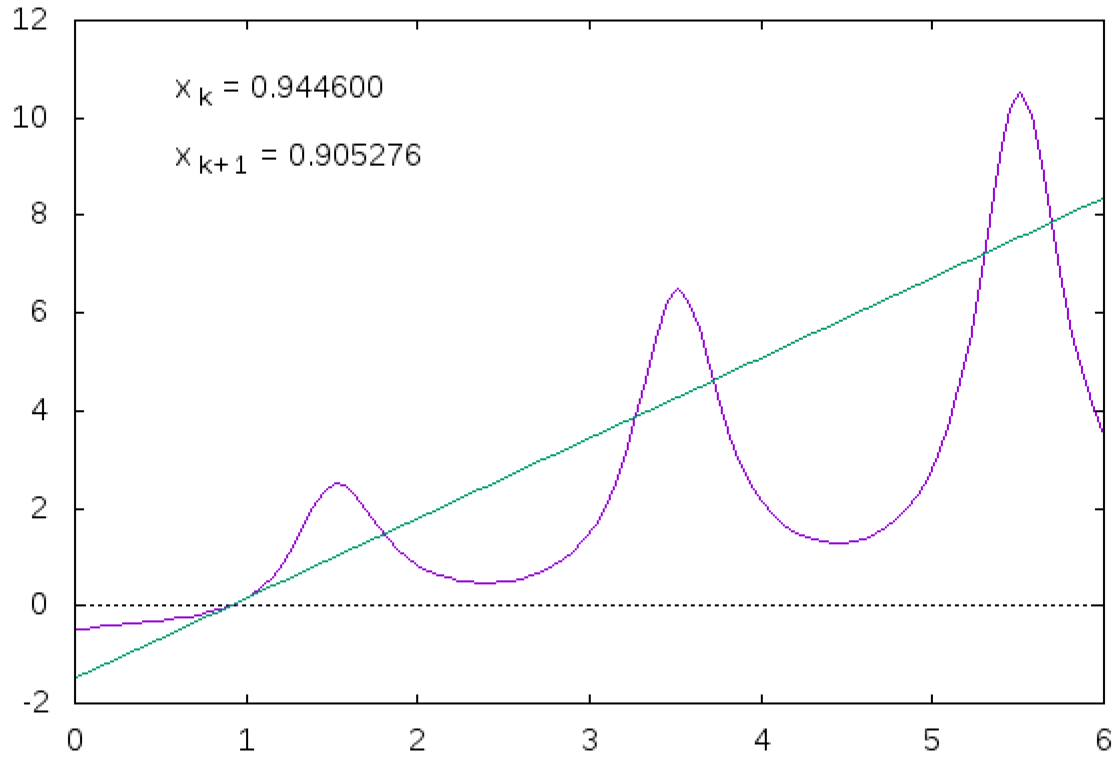
Newton's algorithm

- Second case: x_0=0.5, ε =10^{-4}



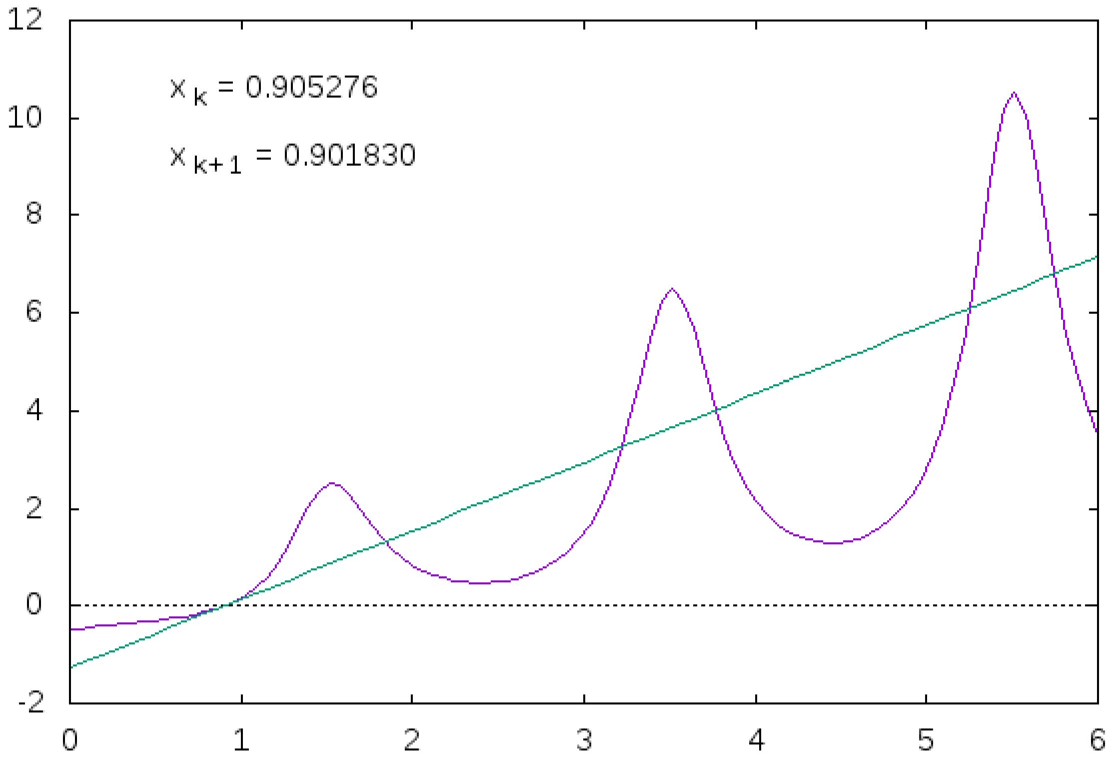
Newton’s algorithm

- Second case: x_0=0.5, ε =10^{-4}



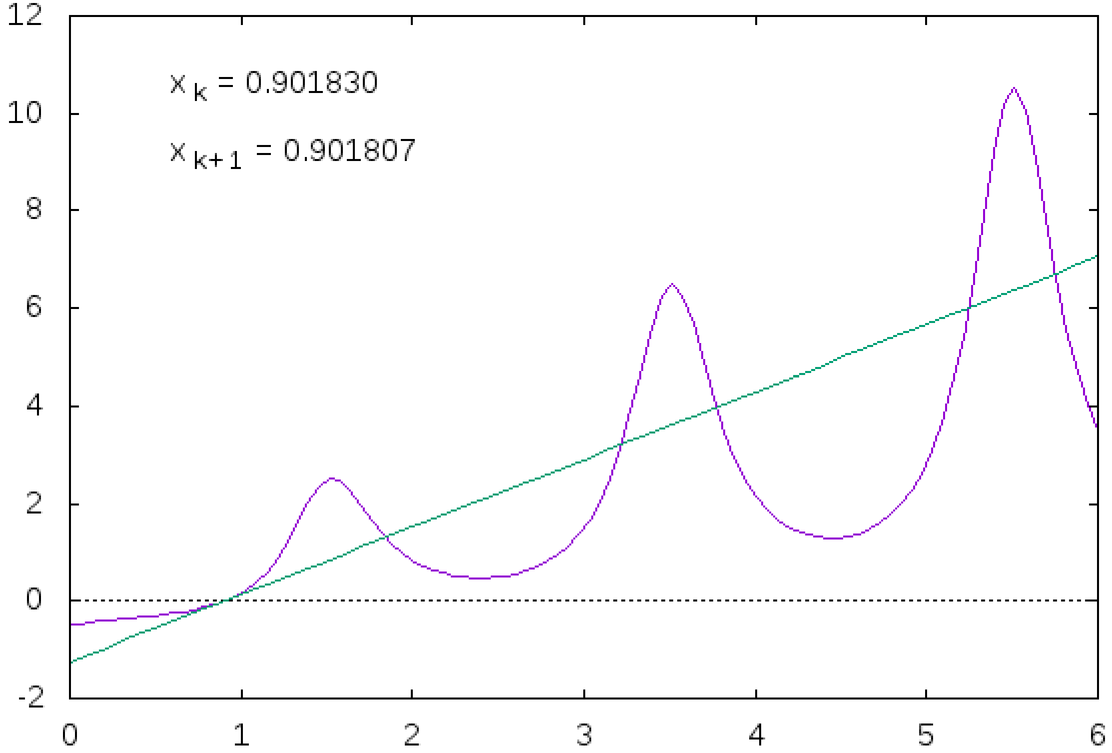
Newton's algorithm

- Second case: x_0=0.5, ε =10^{-4}



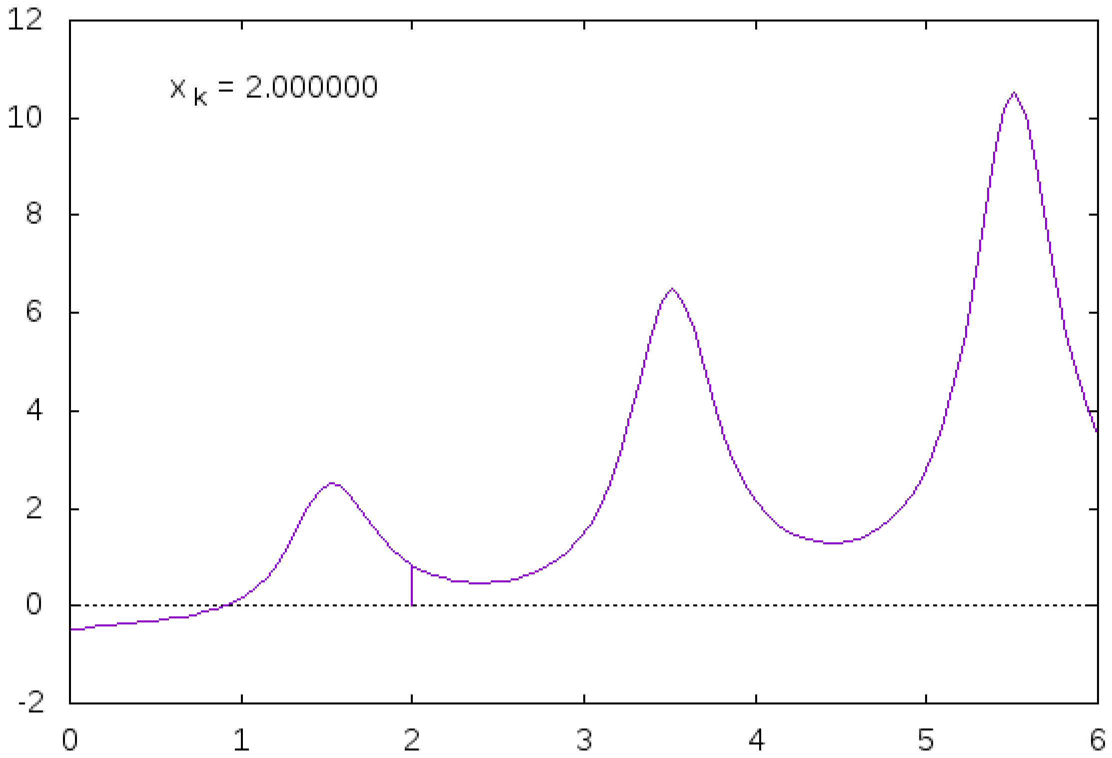
Newton's algorithm

- Second case: x_0=0.5, ε =10^{-4}



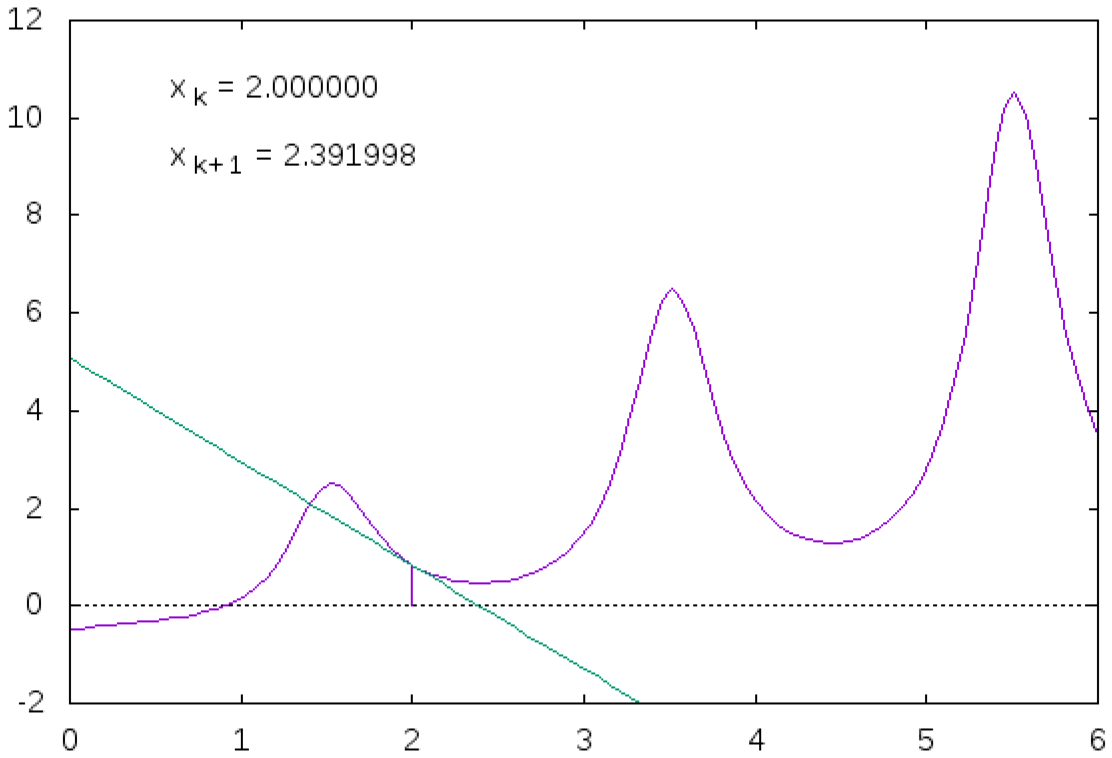
Newton’s algorithm

- Second case: x_0=2.0, ε =10^{-4}



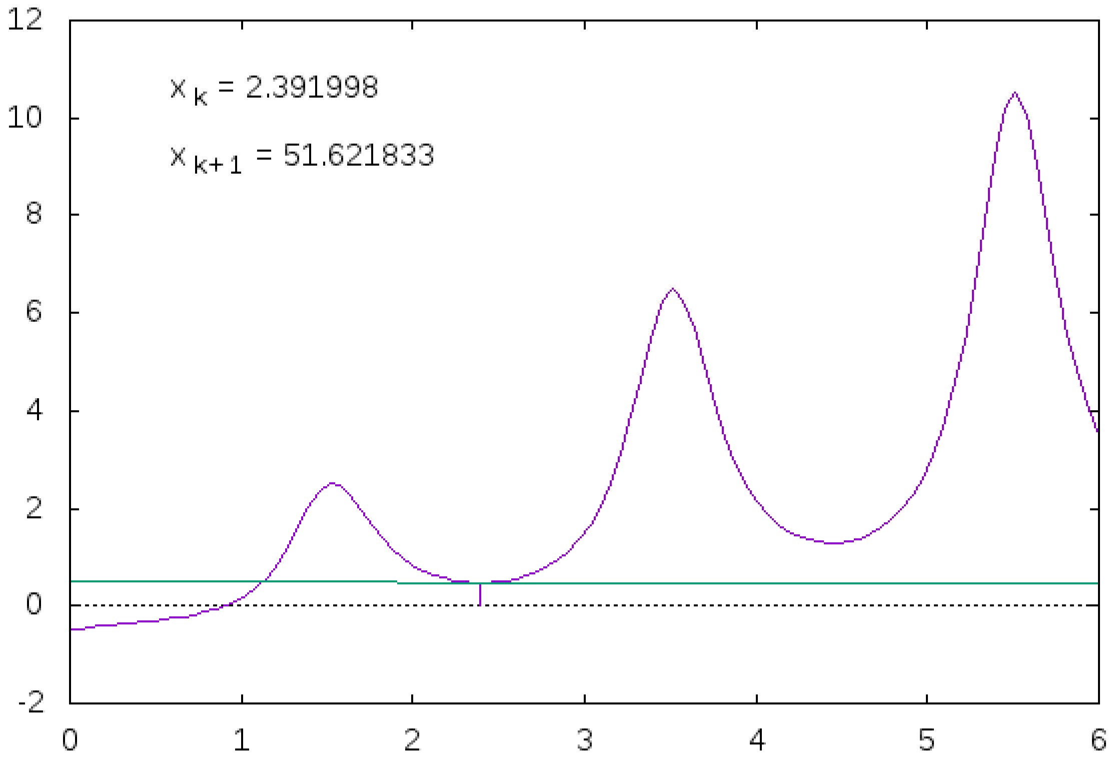
Newton's algorithm

- Second case: x_0=2.0, ε =10^{-4}



Newton's algorithm

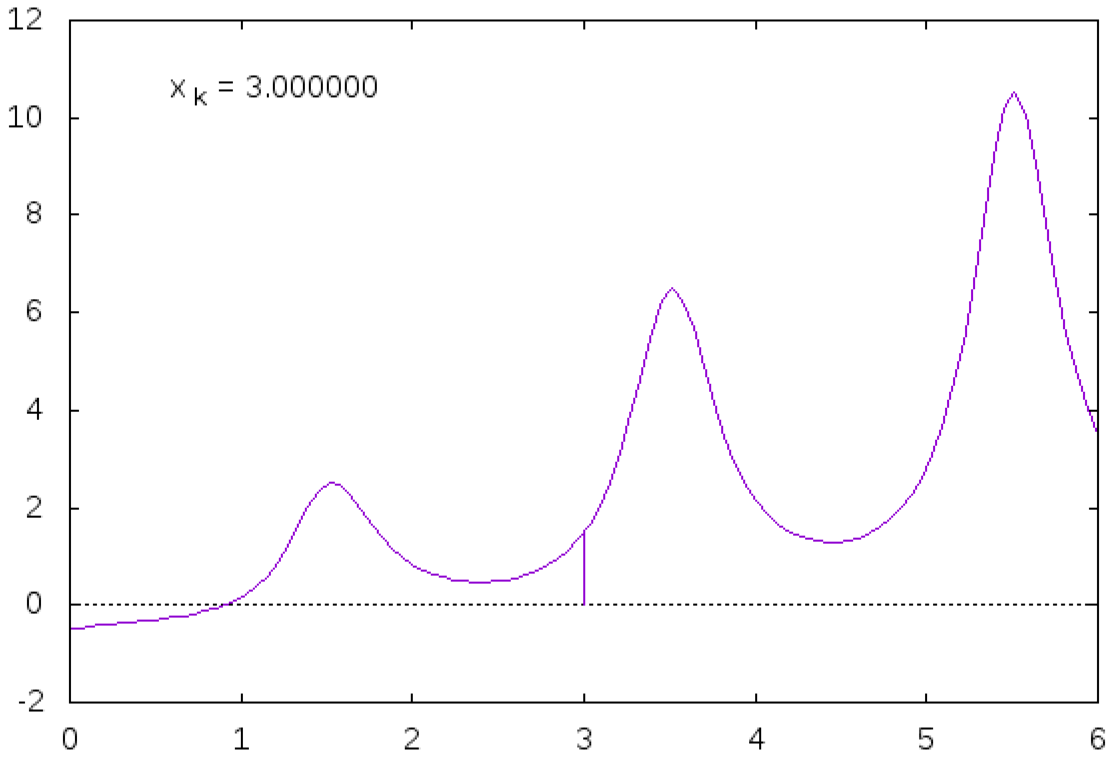
- Second case: x_0=2.0, ε =10^{-4}



No convergence!

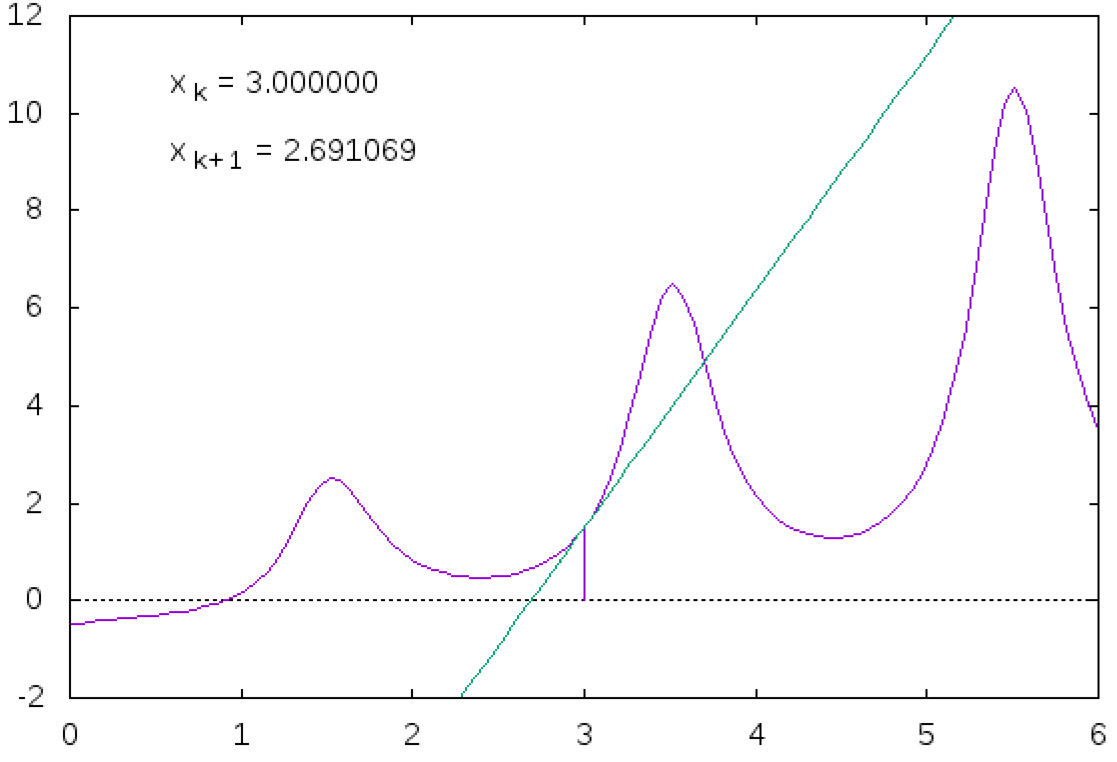
Newton's algorithm

- Second case: x_0=3.0, ε =10^{-4}



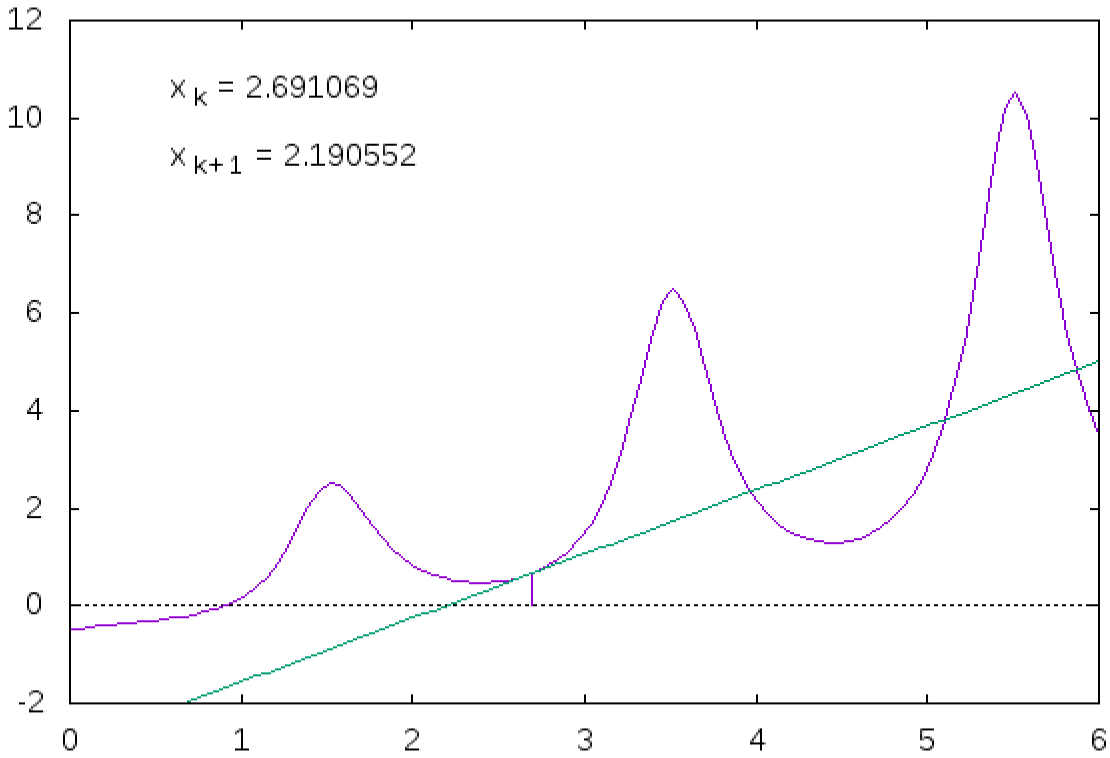
Newton's algorithm

- Second case: x_0=3.0, ε =10^{-4}



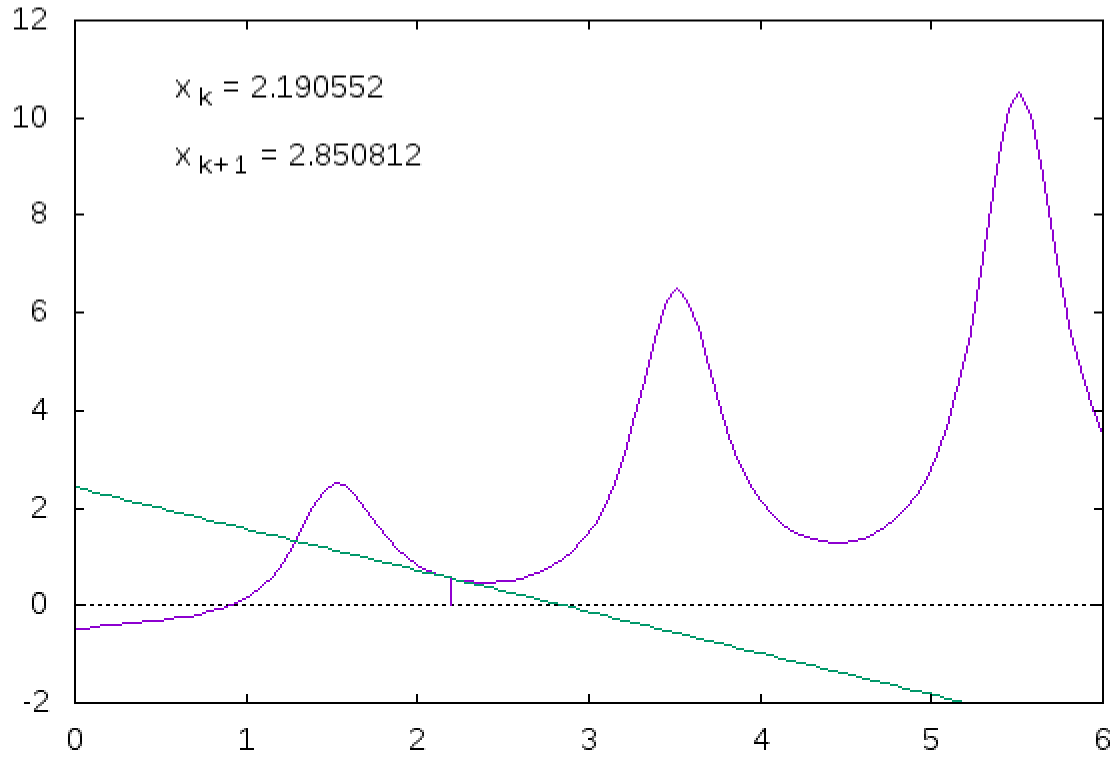
Newton's algorithm

- Second case: x_0=3.0, ε =10^{-4}



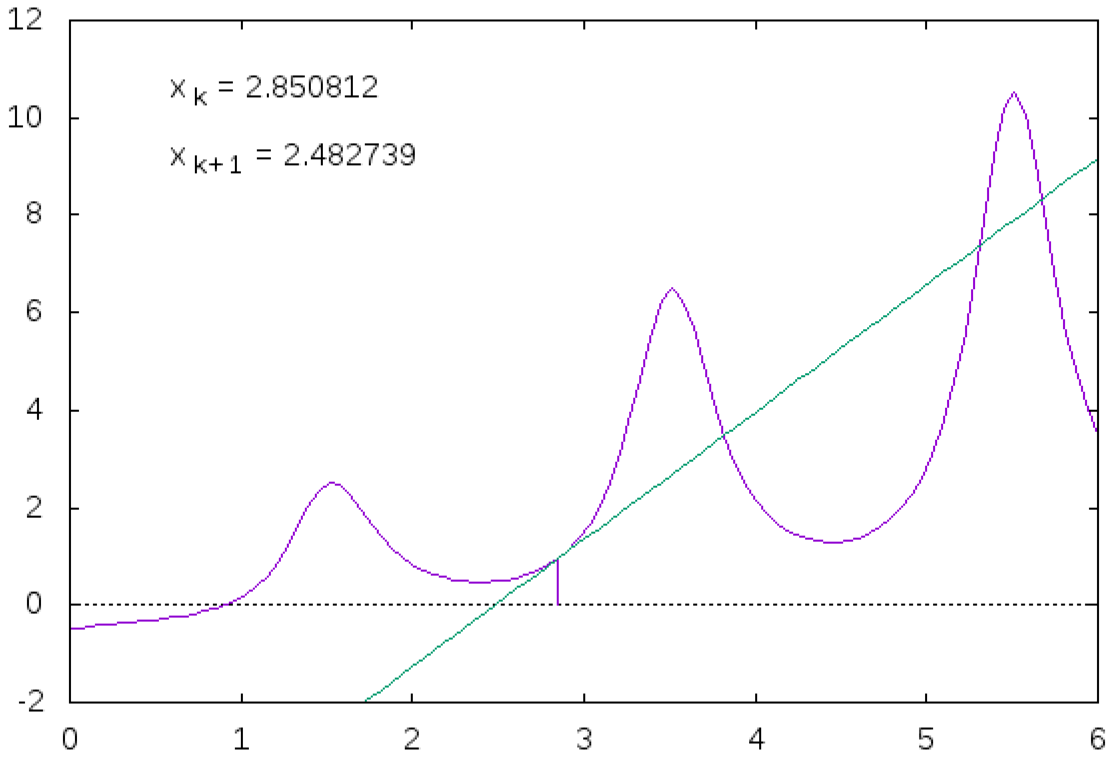
Newton's algorithm

- Second case: x_0=3.0, ε =10^{-4}



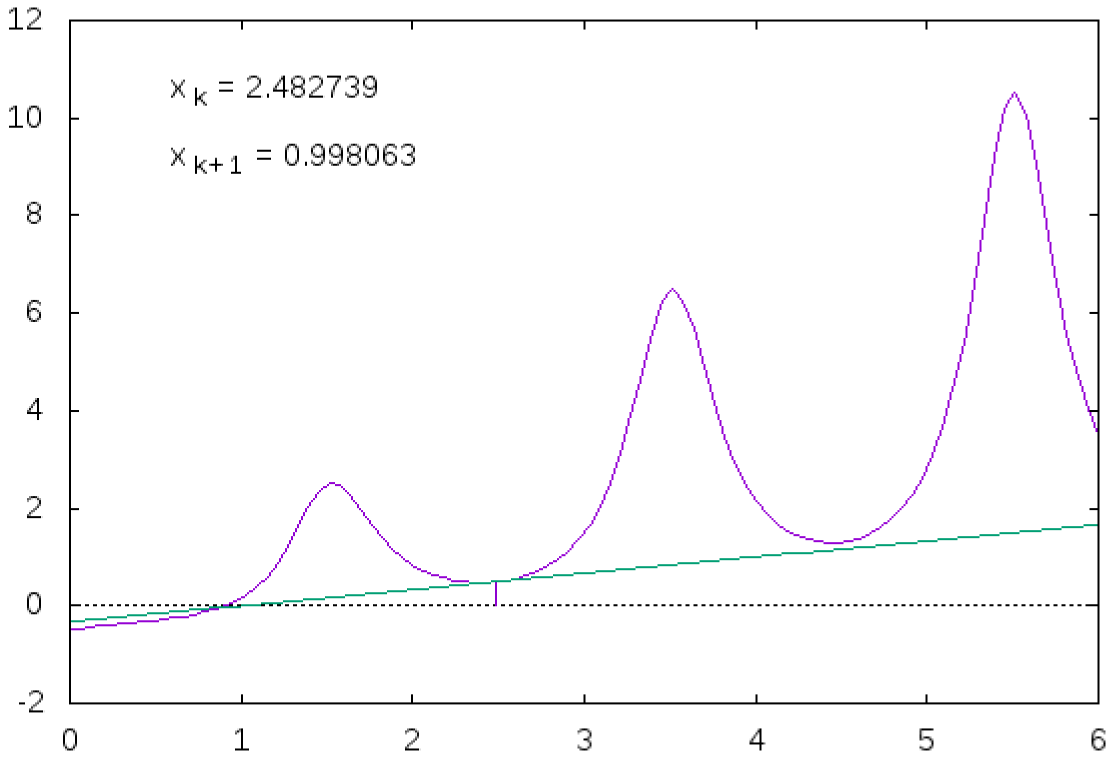
Newton's algorithm

- Second case: x_0=3.0, ε =10^{-4}



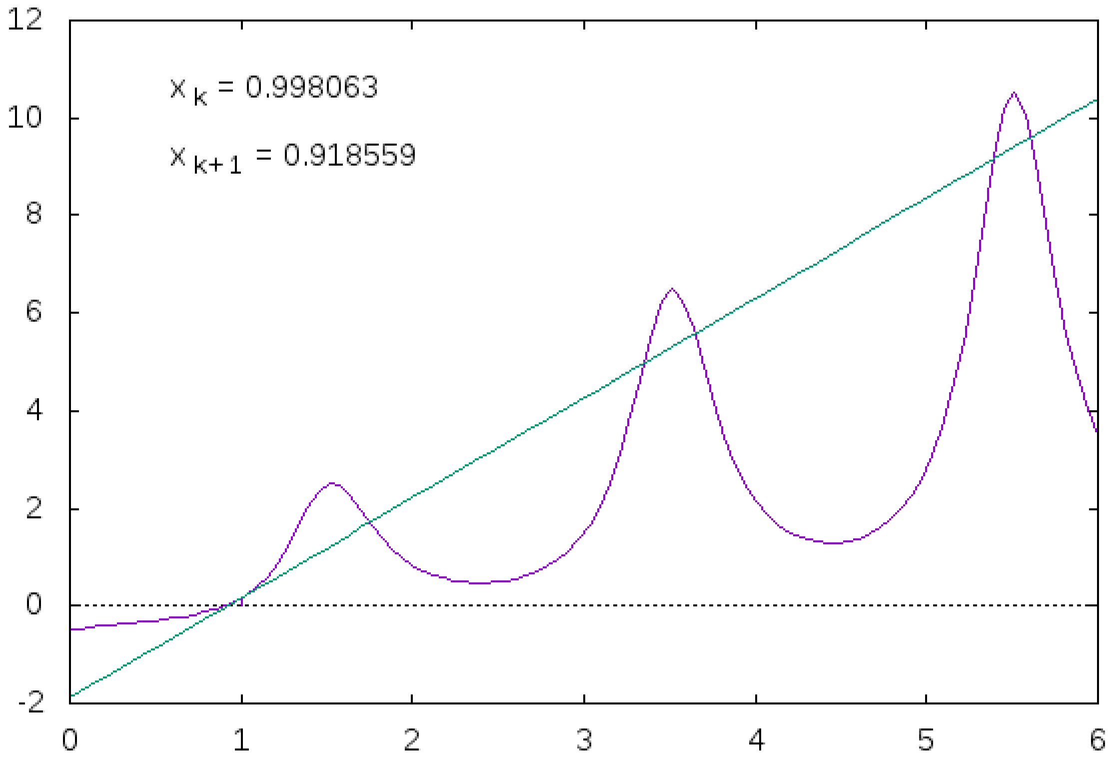
Newton's algorithm

- Second case: x_0=3.0, ε =10^{-4}



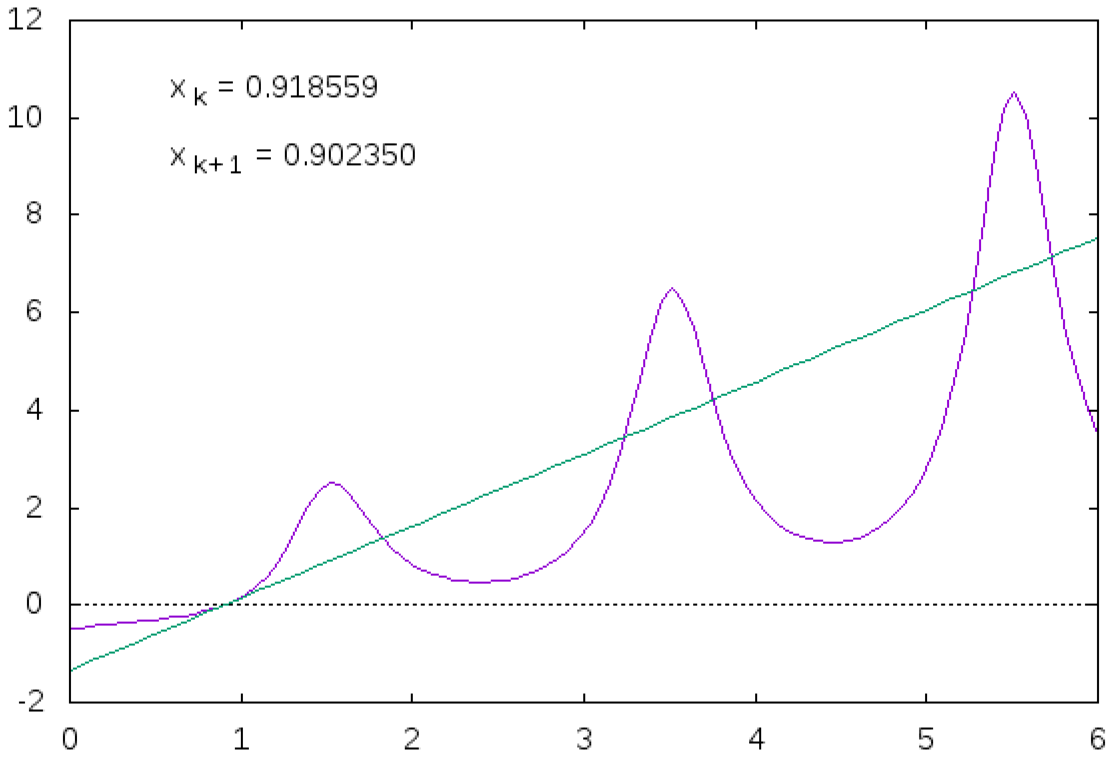
Newton's algorithm

- Second case: x_0=3.0, ε =10^{-4}



Newton's algorithm

- Second case: x_0=3.0, ε =10^{-4}



and so on!...

Newton’s algorithm

- In general, the **best thing to do** is:
  - At first, use a method that always converges, like the bisection method, even if it converges slowly, with a rough precision, in order to limit the interval in which the zero is supposed to exist;
  - Then, use Newton’s method to find a quickly converging solution;
  - In case the second step does not converge, try a different initial guess.

As often happens in life, **a fair amount of “good luck” is fundamental**!

Newton's algorithm in more dimensions

- Quite often, one needs to find the roots of a multi-dimensional system of transcendental equations, like:

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) & = 0 \\ f_2(x_1, x_2, \ldots, x_n) & = 0 \\ \ldots & = 0 \\ f_n(x_1, x_2, \ldots, x_n) & = 0 \end{cases}$$

where f_i are *n* functions:

$$f_i : \mathbb{R}^n \to \mathbb{R}$$

Newton's algorithm in more dimensions

- Just to keep things simple, we can make an analogy with the one-dimensional case.
- We notice that:

$$x^{(k+1)} = x^{(k)} - \frac{f\left[x^{(k)}\right]}{f'\left[x^{(k)}\right]}$$

can be re-written as:

$$f'\left[x^{(k)}\right] \cdot \left[x^{(k+1)} - x^{(k)}\right] = f'\left[x^{(k)}\right] \cdot \delta x^{(k)} = -f\left[x^{(k)}\right]$$

that is, the product of the derivative by the "correction" to the $x^{(k)}$ is equal to $-f\left[x^{(k)}\right]$.

Newton's algorithm in more dimensions

- In n dimensions:
  - The derivative is substituted by the gradient of f ;
  - The scalar $x^{(k)}$ becomes a vector $\bar{x}^{(k)}$ as well as the "correction" $\delta x^{(k)}$ becomes a vector $\delta\bar{x}^{(k)}$;
  - The product becomes a "dot product" between the gradient and the $\delta\bar{x}^{(k)}$.
- That is, for the i-th function f_i we have:

$$\nabla f_i[\bar{x}^{(k)}] \cdot \delta\bar{x}^{(k)} = -f_i[\bar{x}^{(k)}]$$

Newton's algorithm in more dimensions

- Which may be written explicitly as:

$$\left.\frac{\partial f_1}{\partial x_1}\right|_{\bar{x}^{(k)}} \delta x_1^{(k)} + \left.\frac{\partial f_1}{\partial x_2}\right|_{\bar{x}^{(k)}} \delta x_2^{(k)} + \ldots + \left.\frac{\partial f_1}{\partial x_n}\right|_{\bar{x}^{(k)}} \delta x_n^{(k)} = -f_1[\bar{x}^{(k)}]$$

$$\left.\frac{\partial f_2}{\partial x_1}\right|_{\bar{x}^{(k)}} \delta x_1^{(k)} + \left.\frac{\partial f_2}{\partial x_2}\right|_{\bar{x}^{(k)}} \delta x_2^{(k)} + \ldots + \left.\frac{\partial f_2}{\partial x_n}\right|_{\bar{x}^{(k)}} \delta x_n^{(k)} = -f_2[\bar{x}^{(k)}]$$

$$\ldots =$$

$$\left.\frac{\partial f_n}{\partial x_1}\right|_{\bar{x}^{(k)}} \delta x_1^{(k)} + \left.\frac{\partial f_n}{\partial x_2}\right|_{\bar{x}^{(k)}} \delta x_2^{(k)} + \ldots + \left.\frac{\partial f_n}{\partial x_n}\right|_{\bar{x}^{(k)}} \delta x_n^{(k)} = -f_n[\bar{x}^{(k)}]$$

or, in matrix form:

$$J|_{\bar{x}^{(k)}} \delta\bar{x}^{(k)} = -\begin{pmatrix} f_1[\bar{x}^{(k)}] \\ f_2[\bar{x}^{(k)}] \\ \ldots \\ f_n[\bar{x}^{(k)}] \end{pmatrix}$$

Newton's algorithm in more dimensions

- Where J is the Jacobian matrix:

$$\mathrm{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$$

- The "correction" $\delta \bar{x}^{(k)}$ is then given by the solution of a linear system of equations!

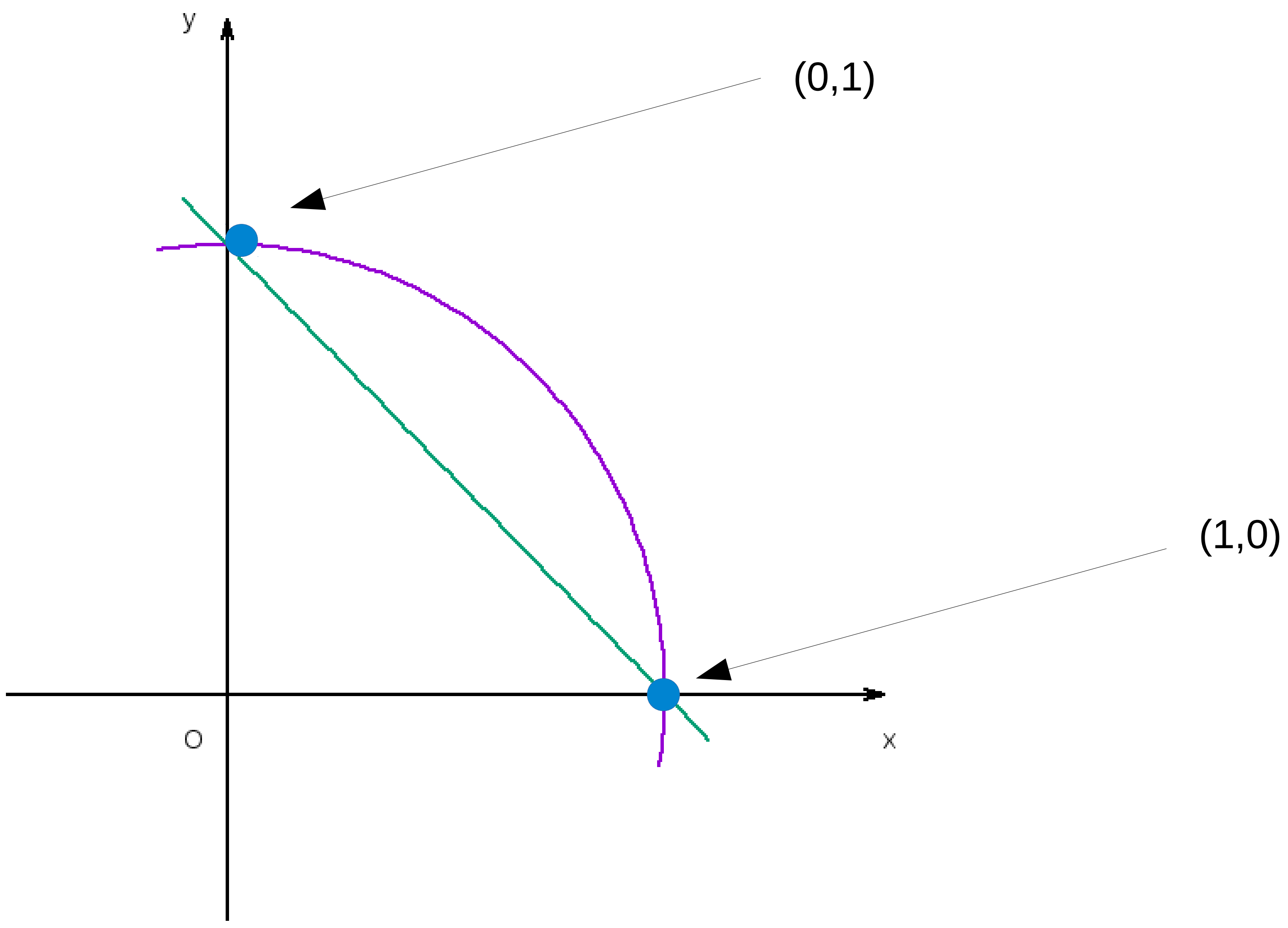
Newton's algorithm in more dimensions

- Example (n = 2!):

$$\begin{cases} f_1(x,y) & = x^2 + y^2 - 1 = 0 \\ f_2(x,y) & = x + y - 1 = 0 \end{cases}$$

- The first equation of the system represents a circle with center in the origin and radius = 1.
- The second equation represents the straight line inclined of $3\pi/4$ with respect to the x axis.
- The solution of such a system is given by the intersections between the two curves:

Newton's algorithm in more dimensions



Newton's algorithm in more dimensions

- The Jacobian matrix, evaluated in $x^{(k)}$, is then given by:

$$J\big|_{\bar{x}^{(k)}} = \begin{pmatrix} 2x & 2y \\ 1 & 1 \end{pmatrix}\Bigg|_{\bar{x}^{(k)}} = \begin{pmatrix} 2x^{(k)} & 2y^{(k)} \\ 1 & 1 \end{pmatrix}$$

therefore the "correction" $\delta\bar{x}^{(k)}$ is given by the solution of the system:

$$\begin{pmatrix} 2x^{(k)} & 2y^{(k)} \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} \delta x^{(k)} \\ \delta y^{(k)} \end{pmatrix} = - \begin{pmatrix} x^2 + y^2 - 1 \\ x + y - 1 \end{pmatrix}\Bigg|_{\bar{x}^{(k)}}$$

$$= - \begin{pmatrix} [x^{(k)}]^2 + [y^{(k)}]^2 - 1 \\ x^{(k)} + y^{(k)} - 1 \end{pmatrix}$$

Newton's algorithm in more dimensions

Where the $x^{(k)}$ and $y^{(k)}$ are known, and therefore we get the "correction" $\delta\bar{x}^{(k)}$ as:

$$\begin{pmatrix} \delta x^{(k)} \\ \delta y^{(k)} \end{pmatrix} = -J^{-1}|_{\bar{x}^{(k)}} \begin{pmatrix} [x^{(k)}]^2 + [y^{(k)}]^2 - 1 \\ x^{(k)} + y^{(k)} - 1 \end{pmatrix}$$

$$= -\frac{1}{x^{(k)} - y^{(k)}} \begin{pmatrix} 1/2 & -y^{(k)} \\ -1/2 & x^{(k)} \end{pmatrix} \begin{pmatrix} [x^{(k)}]^2 + [y^{(k)}]^2 - 1 \\ x^{(k)} + y^{(k)} - 1 \end{pmatrix}$$

and, finally, the next approximation of the solution, given by:

$$\begin{pmatrix} x^{(k+1)} \\ y^{(k+1)} \end{pmatrix} = \begin{pmatrix} x^{(k)} \\ y^{(k)} \end{pmatrix} + \begin{pmatrix} \delta x^{(k)} \\ \delta y^{(k)} \end{pmatrix}$$

Newton's algorithm in more dimensions

- By implementing this in a code, we get, for instance when: $x^{(0)}$=0.3, $y^{(0)}$=0.2, ε =10^-4:

Iter.	$x^{(k+1)}$	$y^{(k+1)}$	$\lvert\delta\bar{x}^{(k)}\rvert$
1	3.65	-2.65	4.398300
2	2.11468	-1.11468	2.17127
3	1.38476	-0.38476	1.03227
4	1.08366	-0.08366	0.42581
5	1.00600	-0.00600	0.10983
6	1.00004	-3.55232×10^{-5}	0.00843
7	1.00000	-1.26181×10^{-9}	5.02356×10^{-5}

which converges to (1.0,0.0)!

Newton's algorithm in more dimensions

- Instead, when: $x^{(0)}$=0.1, $y^{(0)}$=0.3, ε =10^-4:

Iter.	$x^{(k+1)}$	$y^{(k+1)}$	$\lvert\delta\bar{x}^{(k)}\rvert$
1	-1.25	2.25	2.37171
2	-0.44643	1.44643	1.13642
3	-0.10529	1.10529	0.48244
4	-0.00916	1.00916	0.13595
5	-8.23523 $\times 10^{-5}$	1.00008	0.01283
6	-6.78078 $\times 10^{-9}$	1.00000	0.00012
7	-9.31642 $\times 10^{-17}$	1.0	9.58948 $\times 10^{-9}$

which converges to (0.0, 1.0)!