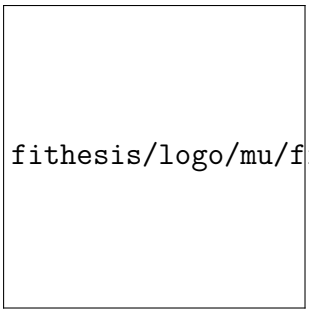
Masaryk University
Faculty of Informatics



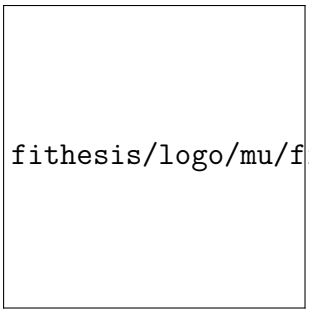
Thesis title

Master's Thesis

Adrian

Brno, Spring 2017

Masaryk University
Faculty of Informatics



Thesis title

Master's Thesis

Adrian

Brno, Spring 2017

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Adrian

Advisor: John Smith

Acknowledgement

This is the acknowledgement for my thesis, which can span multiple paragraphs.

Abstract

This is the abstract of my thesis, which can
span multiple paragraphs.

Keywords

keyword1, keyword2, …

Contents

1 Introduction

Intro

2 Chapter

2.1 Definitions

2.1.1 Univariate polynomial

A univariate polynomial f is a mathematical expression of the form

$$f = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 \tag{2.1}$$

where the $a_n, a_{n-1}, \ldots, a_1, a_0$ are the coefficients of the polynomial, n is any nonnegative integer and x is called an indeterminate or a variable. The highest $n \geq 0$ is called the degree of the polynomial (such n exists, since the set $\{i \mid f_i \neq 0\}$ is finite) and $a_n \neq 0$ is called the leading coefficient. If every a_k, $0 \leq k \leq n$, is a real number, we say that f is polynomial over $\mathbb{R}$ or simply real polynomial.

2.1.2 Roots of a univariate polynomial

Let $f = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ be a real polynomial, $c \in \mathbb{R}$. Then an element $a_n c^n + a_{n-1} c^{n-1} + \ldots + a_1 c + a_0$ is called a value of the polynomial and we denote it as $f(c)$.

Using this, we can create a polynomial function by mapping every element $x \in \mathbb{R}$, to the result of a substitution $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ [2].

Let f be a polynomial over R, $c \in \mathbb{R}$. We say that c is a root of the polynomial f if $f(c) = 0$ [1].

2.2 Approximation of a root using iterative methods

The iterative methods generally require knowledge of one or more initial guesses for the desired root(s) of the polynomial. This often poses a problem itself and there are techniques and methods for finding them. The simplest method for finding a guess is by looking at the plot of the polynomial, which is often not possible (e.g. when dealing with very complex and long polynomials). Some of these methods will be shown later and thus for this section, we will assume we already have a guess.

2.2.1 Bisection method

The simplest method for finding a better approximation to a root is **bisection method**.

Theorem 2.2.1. *Let f be a function on $\mathbb{R}$. Consider an interval $I = [a, b]$ in f such that f is continuous on I and $\lambda \in \mathbb{R}$ such that λ lies between $f(a)$ and $f(b)$. Then there exists a γ, $\gamma \in [a, b]$, such that $f(\gamma) = \lambda$.*

This is called the **intermediate value theorem**[3]. Now, assume a function $f(x)$ that is continuous on interval $[a, b]$ and that $f(a)f(b) < 0$. Then according to the intermediate value theorem there must be at least one root in $[a, b]$. The interval may be chosen large enough that there is more than one root, this is not a problem however, since the algorithm will always converge to some root α in $[a, b]$ and a smaller interval containing only one root. Since all polynomial functions are continuous [4], we can use this theorem to create an algorithm.

Algorithm 1 Bisection algorithm

Precondition: f polynomial function, a, b interval bounds, ϵ precision error

```
1: function Bisection(f, a, b, ε)
2:     c ← (a+b)/2
3:     if c − a ≤ ε then
4:         return c
5:     if f(a) ∗ f(c) < 0 then
6:         return Bisection(f, a, c, ε)
7:     else
8:         return Bisection(f, c, b, ε)
```

Example 2.2.1. Find a root α of

$$2x^4 - 3x - 2 \tag{2.2}$$

with the precision $\epsilon = 0.00005$.

It is fairly straightforward to show that there is a root located between $1 < \alpha < 2$, so we will use this interval as our initial guess. The results are shown in the table below.

Iteration	c_n	$f(c_n)$
1	1.50000	3.62500
2	1.25000	-0.86719
3	1.37500	1.023931
4	1.31250	-0.00241
5	1.34375	0.48960
6	1.32813	0.23842
7	1.32031	0.11674
8	1.31641	0.05685
9	1.31445	0.02715
10	1.31348	0.01235
11	1.31299	0.00497
12	1.31275	0.00129
13	1.31262	-0.00055
14	1.31268	0.00037
15	1.31265	-0.00009

Table 2.1: Bisection algorithm on example

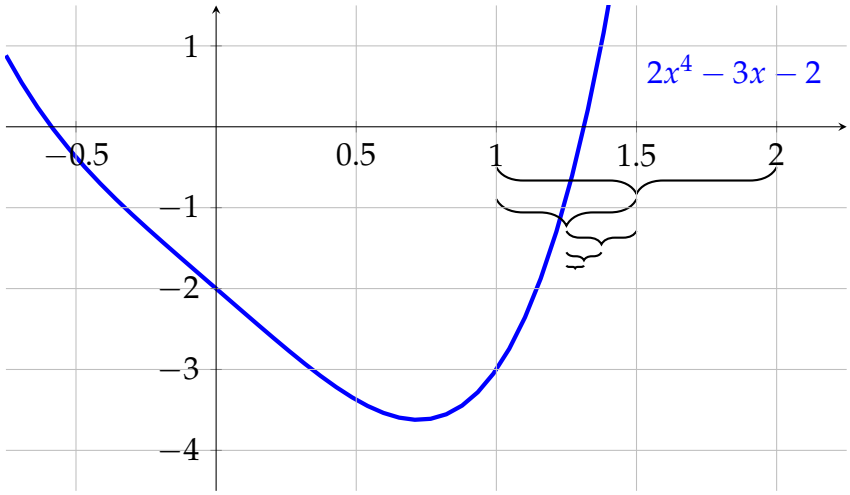


Figure 2.1: First 5 iterations of bisection algorithm on the example

The correct approximation is

$$\alpha \doteq 1.31265975467417 \tag{2.3}$$

The error of the final iteration is

$$|\alpha - c_{15}| \doteq 0.00000975467417 \tag{2.4}$$

which is smaller than the required error bound (0.00005). Upon closer inspection, it can be noticed that the algorithm found a solution with enough precision in an earlier iteration already (for example c_{13}) and it may seem as the computation could have been stopped right then. However, this fact was not known beforehand, because there is no possibility to predict the accuracy in an earlier iteration during computation.

While every iteration gives a better approximation of the true solution, the algorithm is converging rather slowly. To examine the speed of convergence, we first need to characterize it.

Definition 1. A sequence of iterates $\{x_n | n \geq 0\}$ is said to converge with *order* $p \geq 1$ to a point α if

$$|\alpha - x_{n+1}| \leq c|\alpha - x_n|^p \qquad n \geq 0$$

for some $c > 0$. If $p = 1$, the sequence is said to *converge linearly* to α. In that case, we require $c < 1$; the constant c is called the *rate of linear convergence* of x_n to α [5, p. 56].

Let c_n denote the nth value of c in the algorithm. Then

$$\alpha = \lim_{n \to \infty} c_n \tag{2.5}$$

$$|\alpha - c_n| \leq \left[\frac{1}{2}\right]^n (b - a) \tag{2.6}$$

where $b - a$ denotes the length of the original interval input into the algorithm. Using our definition, we can say that the bisection algorithm has linear convergence with a rate of $\frac{1}{2}$. That does not necessarily mean that in every iteration the actual error decreases by a factor of

$\frac{1}{2}$, but that the *average* rate of decrease is $\frac{1}{2}$. This means that it takes on average approximately 3.32 iterations ($\log_2 10$) to compute a single digit.

The major drawback of this algorithm is its very slow rate of convergence, specifically compared to other methods described in the following sections.

On the other hand, the *Bisection algorithm* has several advantages. The first of them being that it is guaranteed to converge if the prerequisites are met(i.e. the f is continuous - which is true for all polynomial functions - and $f(a)f(b) < 0$). The second one being a reasonable error bound. This method also provides upper and lower bounds on the root α in every iteration and such belongs in class of methods called *enclosure methods* [5].

2.2.2 Newton's method

Newton's method (sometimes also called **Newton-Raphson method**), named after Isaac Newton and Joseph Raphson, is another method for finding better approximations to a root of a real polynomial function (or in general, any real-valued function). The basic idea of the method is based on the fact that given a starting point x_0, that is sufficiently close to the root, one can approximate the function by computing its tangent line at the point $(x_0, f(x_0))$. Then, one can use the x-intercept of the tangent line, which typically provides a better approximation, and repeat this process ad infinitum [5] (or until sufficient precision is reached).

Assume that real-valued function $f(x)$ is differentiable on interval $[a, b]$ and that we have an initial approximation x_0. Then, using calculus, we can derive the formula for a better approximation x_1.

To compute the better approximation x_1, we need to use the formula that describes the slope m of our tangent line

$$m = \frac{f(x_1) - f(x_0)}{x_1 - x_0}. \tag{2.7}$$

Substituting $f(x_1)$ by 0 and manipulating the equation, we obtain

$$x_1 = x_0 - \frac{f(x_0)}{m}. \tag{2.8}$$

And since the slope m of the tangent line is the derivative of the function f at the point x_0, we obtain our desired formula

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}. \tag{2.9}$$

The general formula is obtained by iterating the process, by replacing x_0 with x_1, ad infinitum, which gives us

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{2.10}$$

Since all polynomial functions are differentiable, we can apply this formula to our problem. Newton's method is exceptionally powerful and one of the most well known techniques for finding the roots, since its rather easy to implement and converges very quickly.

Theorem 2.2.2 ([5, p. 60])**.** *Assume that $f(x), f'(x)$, and $f''(x)$ are all continuous for every x in some neighbourhood $I = [\alpha - \epsilon, \alpha + \epsilon]$ of α, α is a root of f, and $f'(x) \neq 0, \forall x \in I$. Then if x_0 is chosen sufficiently close to α, the iterates $x_n, n \geq 0$ of* (2.10) *will converge to α. Furthermore,*

$$\lim_{n\to\infty} \frac{\alpha - x_{n+1}}{(\alpha - x_n)^2} = \frac{-f''(\alpha)}{2f'(\alpha)} \tag{2.11}$$

proving that the convergence is quadratic.

Proof. (Skecth) First, since f is twice continuously differentiable around the root α, we can use a Taylor series expansion[5, p. 59] to second order about a point close to α, x_n, to represent $f(\alpha)$. The expansion of $f(\alpha)$ about x_n is

$$f(\alpha) = f(x_n) + f'(x_n)(\alpha - x_n) + \frac{f''(\xi_n)}{2!}(\alpha - x_n)^2 \tag{2.12}$$

where ξ_n is between x and α. Since α is root, we can replace $f(\alpha)$ with 0 and by manipulating the equation we get

$$-\alpha + x_n - \frac{f(x_n)}{f'(x_n)} = \frac{f''(\xi_n)}{2f'(x_n)}(\alpha - x_n)^2 \tag{2.13}$$

where we can use the definition of x_{n+1} from (2.10) and multiply the equation by -1 to get

$$\alpha - x_{n+1} = \frac{-f''(\xi_n)}{2f'(x_n)}(\alpha - x_n)^2 \tag{2.14}$$

Taking the absolute value of both sides of the equation we get

$$|\alpha - x_{n+1}| = \frac{|f''(\xi_n)|}{|2f'(x_n)|}\,|\alpha - x_n|^2 \tag{2.15}$$

Now we pick the sufficiently small interval $I = [\alpha - \epsilon, \alpha + \epsilon]$ on which $f'(x) \neq 0$ (which exists since $f'(x)$ is continuous) and then let

$$M = sup_{x \in I} \frac{|f''(x)|}{|2f'(x)|} \tag{2.16}$$

Pick x_0, such that $|\alpha - x_0| \leq \epsilon$ and $M\,|\alpha - x_0| < 1$[5, p. 61]. Then $M\,|\alpha - x_1| < 1$ and $M\,|\alpha - x_1| \leq M\,|\alpha - x_0|$. Using induction we can apply this argument for every $n \geq 1$, showing that $|\alpha - x_n| \leq \epsilon$ and $M\,|\alpha - x_n| < 1$ for all $n \geq 1$. This, in combination with (2.15), gives us

$$|\alpha - x_{n+1}| \leq M\,|\alpha - x_n|^2 \tag{2.17}$$

$$M\,|\alpha - x_{n+1}| \leq (M\,|\alpha - x_n|)^2 \tag{2.18}$$

and

$$M\,|\alpha - x_n| \leq (M\,|\alpha - x_0|)^{2^n} \tag{2.19}$$

$$M\,|\alpha - x_n| \leq \frac{1}{M}(M\,|\alpha - x_0|)^{2^n} \tag{2.20}$$

Since $M\,|\alpha - x_0| < 1$, this shows that $x_n \to \alpha$ as $n \to \infty$. The aforementioned point ξ_n is between x_n and α, which implies that $\xi_n \to \alpha$ as $n \to \infty$. Thus [5, p. 61]

$$\lim_{n\to\infty} \frac{\alpha - x_{n+1}}{(\alpha - x_n)^2} = -\lim_{n\to\infty} \frac{f''(\xi_n)}{2f'(x_n)} = \frac{-f''(\alpha)}{2f'(\alpha)} \tag{2.21}$$

□

However, this method has several deficiencies.

Firstly, just by observing the equation, we can see that if the iteration point is stationary, then x_{n+1} is undefined, since $f'(x_n) = 0$ (which means the tangent line is parallel to the x-axis).

Secondly, for some polynomial functions, the method may enter an infinite cycle. This happens if, for example, x_1 produces $x_2 = x_0$ as output, causing the method to alternate between these two results infinitely.

Thirdly, if the initial guess is not close enough or the first derivative does not behave well in the neighbourhood of a specific root, the method may skip this root as the tangent line will intercept the x-axis close to another root. This may pose a problem, if someone is looking for a root located specifically in an interval $[a, b]$, but does not pose a problem if one is simply looking for any root of a polynomial function.

Lastly, if the root of the polynomial function has a multiplicity greater than one(i.e. the first derivative of the root is also zero), then the convergence to this root is only linear.

Proof. Let $f(x)$ be a polynomial function with a root α of multiplicity m, $m > 1$, i.e. $f(x) = (x - \alpha)^m g(x)$. Assume that x_0 is sufficiently close to α. Then

$$x_{n+1} = x_n - \frac{(x_n - \alpha)^m g(x_n)}{m(x_n - \alpha)^{m-1} g(x_n) + (x_n - \alpha)^m g'(x_n)} \tag{2.22}$$

$$= x_n - \frac{(x_n - \alpha) g(x_n)}{m g(x_n) + (x_n - \alpha) g'(x_n)} \tag{2.23}$$

$$= \frac{x_n(m g(x_n) + (x_n - \alpha) g'(x_n)) - (x_n - \alpha) g(x_n)}{m g(x_n) + (x_n - \alpha) g'(x_n)} \tag{2.24}$$

$$= \frac{x_n(m-1) g(x_n) + x_n(x_n - \alpha) g'(x_n) + \alpha g(x_n)}{m g(x_n) + (x_n - \alpha) g'(x_n)} \tag{2.25}$$

which, as we get closer to the root, i.e. $x_n \doteq \alpha$, gives us

$$x_{n+1} \doteq x_n \frac{(m-1)}{m} + \frac{\alpha}{m} \tag{2.26}$$

or put differently

$$x_{n+1} \doteq (x_n - \alpha) \frac{(m-1)}{m} + \alpha \tag{2.27}$$

from which we can conclude

$$|\alpha - x_{n+1}| \leq \frac{(m-1)}{m} |\alpha - x_n| \tag{2.28}$$

which by the definition 1 proves the linear convergence. □

Error bounds To estimate the error and provide error bounds of our computation, we will need to use the variation of the **mean value theorem**.

Mean value theorem

Theorem 2.2.3. *Let f be a real-valued polynomial function and a, b real numbers, such that $a < b$. Then there exists some $c \in (a, b)$ such that*

$$f'(c) = \frac{f(b) - f(a)}{b - a}. \tag{2.29}$$

Roughly speaking, it states that given a curve, which starts at point a and ends in point b, there is at least one point at which the tangent to the arc is parallel to the secant connecting the two points. Using this theorem on a polynomial function f with a root α and an approximation x_n, $\alpha < x_n$ we get

$$f'(\xi_n) = \frac{f(x_n - f(\alpha))}{x_n - \alpha} \tag{2.30}$$

where ξ_n being between α and x_n. Since α is the root of f, then we can remove $f(\alpha)$ and manipulating we get

$$\alpha - x_n = \frac{-f(x_n)}{f'(\xi_n)}. \tag{2.31}$$

If $x_n < \alpha$, we arrive to the same conclusion. If $f'(x)$ is not changing rapidly between x_n and α, then $f'(\xi_n) \doteq f'(x_n)$. Combining this with the definition of Newton's method we get

$$\alpha - x_n \doteq \frac{-f(x_n)}{f'(x_n)} = x_{n+1} - x_n. \tag{2.32}$$

This gives us the absolute error estimate

$$\alpha - x_n \doteq x_{n+1} - x_n \tag{2.33}$$

and relative error estimate

$$\frac{\alpha - x_n}{\alpha} \doteq \frac{x_{n+1} - x_n}{x_{n+1}}. \tag{2.34}$$

The Newton algorithm Using the Newton formula (2.10) and the error estimates (2.33), (2.34) provided above, we can create the following algorithm.

Algorithm 2 Newton's algorithm

Precondition: f polynomial function, f' derivative of f, a, b interval bounds, ϵ precision error, max number of maximum iterations, err error flag

```
1:  function NEWTON(f, f', a, b, ε, max)
2:      err ← 1
3:      x_0 ← (a+b)/2
4:      for i ← 1, max do
5:          denom ← f'(x_0)
6:          if denom = 0 then
7:              err ← 2
8:              return err, x_1
9:          x_1 ← x_0 − f(x_0)/denom
10:         if (|(x_1 − x_0)/x_1| ≤ ε) and (|x_1 − x_0| ≤ ε) then
11:             err ← 0
12:             return success, x_1
13:         if (x_1 < a) or (x_1 > b) then
14:             err ← 3
15:             return err, x_1
16:         x_0 ← x_1
17:     return err, x_1
```

The max variable serves as a countermeasure against the case when the method oscillates between two points and does not converge (meaning there is no root located in the interval (a, b)). However, in the case

that the function iterates through the maximum amount of iterations max and returns no root (i.e. $i = max$ and $success$ is false), but one knows the root of the f is located in (a, b), one may try to increase the max or narrow the interval (a, b) (e.g. using bisection) and try running the algorithm again.

The lines 12 to 13 serve as a prevention against the case of overshooting the root. These lines may be omitted if one does not have particular interest in finding the root from the specific interval. As in previous case, if the algorithm ends prematurely because of the latest iteration being outside of the interval, but one is certain that the root is located inside the interval, one may try narrowing the interval (thus giving a better starting point).

The termination condition is a combination of the error bounds, both absolute and relative. The reason for this choice is that while relative error works well in small magnitude and worse when the root is large in magnitude, the opposite is true for absolute error. That is because for error $\epsilon = 10^{-e}$ the relative error gives at least $e - 1$ correct digits. This means that for an approximation in the floating point representation at least $e - 1$ digits of the significand are correct. Whereas the absolute error provides at least $e - 1$ correct digits after the decimal point, when the number is not in the floating point representation.

E.g. Let $f(x) = x^3 - 100x^2 + x - 100$, which has root $\alpha = 100$ and set $\epsilon = 0.1$. Then, using only relative error bound and setting $a = 0, b = 2000$, the algorithm returns as a result $x = 101.5$ after 7 iterations, ending too early. Using absolute error bound, the algorithm ends after 9 iterations with the desired result $x = 100.0$. Now let $f(x) = x^3 - 0.0001x^2 + x - 0.0001$ which has root $\alpha = 0.0001$ and set $\epsilon = 0.00001$. Then, using only absolute error bound and setting $a = 0, b = 100$, the algorithm returns as a result $= x0.000104$ after 13 iterations. Then, using only relative error bound and setting $a = 0, b = 100$, the algorithm returns as a result $x = 0.0001000$ after 14 iterations. While both results are within the precision error, the relative error gives more precise result. Thus the best results are achieved combining both bounds.

Example 2.2.2. Find a root α of

$$2x^4 - 3x - 2 \tag{2.35}$$

with the precision $\epsilon = 0.00005$.

The initial interval is same as in Bisection algorithm to make results comparable.

Iteration	c_n	$f(c_n)$
1	1.50000	3.62500
2	1.34896	0.57565
3	1.31436	0.02569
4	1.31266	0.00006

Table 2.2: Newton's algorithm on example

While the algorithm solves some of it's deficiencies mentioned before, some remain. The method might sometimes fail (when the first derivative of x_n is zero) and the convergence for the roots of higher multiplicity is only linear.

On the other hand, as we can see from the table, the algorithm converges much faster than the previous method, finding the sufficient approximation in only 4 iterations. This makes Newton's method superior to the bisection method in most of the cases.

Bibliography

1. ROSICKÝ, Jiří. *Algebra*. 4. vyd. Brno: Masarykova univerzita, 2007. ISBN 978-80-210-2964-4.
2. LEUNG, K.T.; MOK, I.A.C.; SUEN, S.N. *Polynomials and Equations*. Hong Kong: Hong Kong University Press, 1992. ISBN 962-209-271-3.
3. BINMORE, K.G. *Mathematical Analysis: A Straightforward Approach*. Cambridge: Cambridge University Press, 1997. ISBN 9780521288828.
4. *Polynomials are continuous functions* [online]. Berkeley: The University of California, Berkeley, 2014 [visited on 2017-03-03]. Available from: `https://math.berkeley.edu/~kmill/math1afa2014/poly.pdf`.
5. ATKINSON, Kendall E. *An Introduction to Numerical Analysis*. 2nd ed. Iowa City: John Wiley & Sons, 1989. ISBN 0-471-62489-6.

A An appendix

Here you can insert the appendices of your thesis.