Project 1

Numerical methods, Faculty of Mathematics and Information Science, Warsaw University of Technology

Paweł Pozorski

26.11.2023

Finding zero points of polynomial $w_n(x)$ using Halley's method

Given the polynomial

$$w_n(x) = \sum_{k=0}^{n} a_k T_k(x) T_{n-k}(x)$$

where $x \in \mathbb{R}$, $T_0, T_1, \ldots, T_n$ - 1'st type Chebyshev polynomials ($T_k(x) = \cos(k \arccos(x))$, defined for $x \in [-1, 1]$ and $n = 0, 1, \ldots$), $a_k \in \mathbb{R}$, $n \in \mathbb{N}$ - known constants, we'll be looking for zero points of $w_n(x)$ using Hallye's method.

1. Definition of Hallye's method

Let $f : \mathbb{R} \to \mathbb{R}$ - function of class $C^2(\mathbb{R})$ (so it has continous second derevative), and $x_0 \in \mathbb{R}$ - given starting approximation for f zero point. Using Taylor series of function f for the area of point x_k (where $x_k \in \mathbb{R}$ and $k \in \mathbb{N}$) we get p - the approximation of function f:

$$p(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2} f''(x_k)(x - x_k)^2$$

Assuming $f(x_{k+1}) = 0$ we get

$$f(x_k) + f'(x_k)(x_{k+1} - x_k) + \frac{1}{2} f''(x_k)(x_{k+1} - x_k)^2 = 0$$

Therefore

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k) + \frac{1}{2} f''(x_k)(x_{k+1} - x_k)}$$

Furthermore, values on the right side of this equasion we can approximate with the Newton's method (so $x = x - \frac{f(x_k)}{f'(x_k)}$). This gives us the final formula:

$$x_{k+1} = x_k - \frac{2f(x_k)f'(x_k)}{2(f'(x_k))^2 - f(x_k)f''(x_k)}$$

By creating a sequence $(x_0, x_1, x_2, x_3, \ldots)$ of real points fulfilling upper formula we might be able to create a sequence that is convergent to zero point of f.
So the Halley's method for next approximation takes a point $x_k \in \mathbb{R}$ which is a zero point of a hyperbole that approximates f at point x_k.

2. Computation Limitation

For obvious reasons computers are unable to calculate this limit, therefore some convergence criteria must be introduced. In this work's implementation we're using below condition to stop computation:

$$|x_{k+1} - x_k| < \epsilon \wedge |f(x_{k+1})| < \epsilon$$

for some $\epsilon > 0$. Therefore, this ϵ is maximum error we can get from this computation, which leaves nearly no space for analysing this method errors.

3. Implementation details

Alonside some utility functions to create below plots, implementation in Matlab consists of the following functions:

```
function root = halley(F, x0, tolerance, max_iterations)
    % Halley's method for finding a root of the function F
    % it will be the last element of root return variable
    % earlier entries are how x values in (i-1)th iteration

    % F: function of one parameter x that returns [F(x), F'(x), F''(x)]
    % x0: starting approximation
    % tolerance: convergence tolerance
    % max_iterations: maximum number of iterations

function [Wx, Wdx, Wddx] = W(a, n, x)
    % calculates value of a function W given in task at points x,
    % its first and second derevative

    % a: vector coefficients of the chebyshev polynomial (a_0, a_1, ..., a_n)
    % n: degree of the chebyshev polynomial
    % x: vector of points where the function should be evaluated

function W_func = create_W_func(a)
    % Creates W_n function according to halley function requirements,
    % so the function of just one parameter x that returns W_n(x), W'_n(x), W''_
```

`halley` is responsible for conducting the Halley algorithm until it coverages or exceedes maximum allowed interation count. `W` is our $w_n(x)$ polynomial implementation. For performance reasons, `W` returns all $w_n(x), w_n'(x), w_n''(x)$ at one and calculates the Chebyshev polynomials using dynamic programming and recursive relationship fulfilled by Chebyshev polynomials which goes as follows:

- $T_0(x) = 1$
- $T_1(x) = x$
- $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ for $k = 2, 3, \ldots$.

`W_func` allows easier calls on `W` with already set sequence $a = (a_0, a_1, a_2, a_3, \ldots, a_n)$ and $n \in \mathbb{N}$. Let us see example execution call:

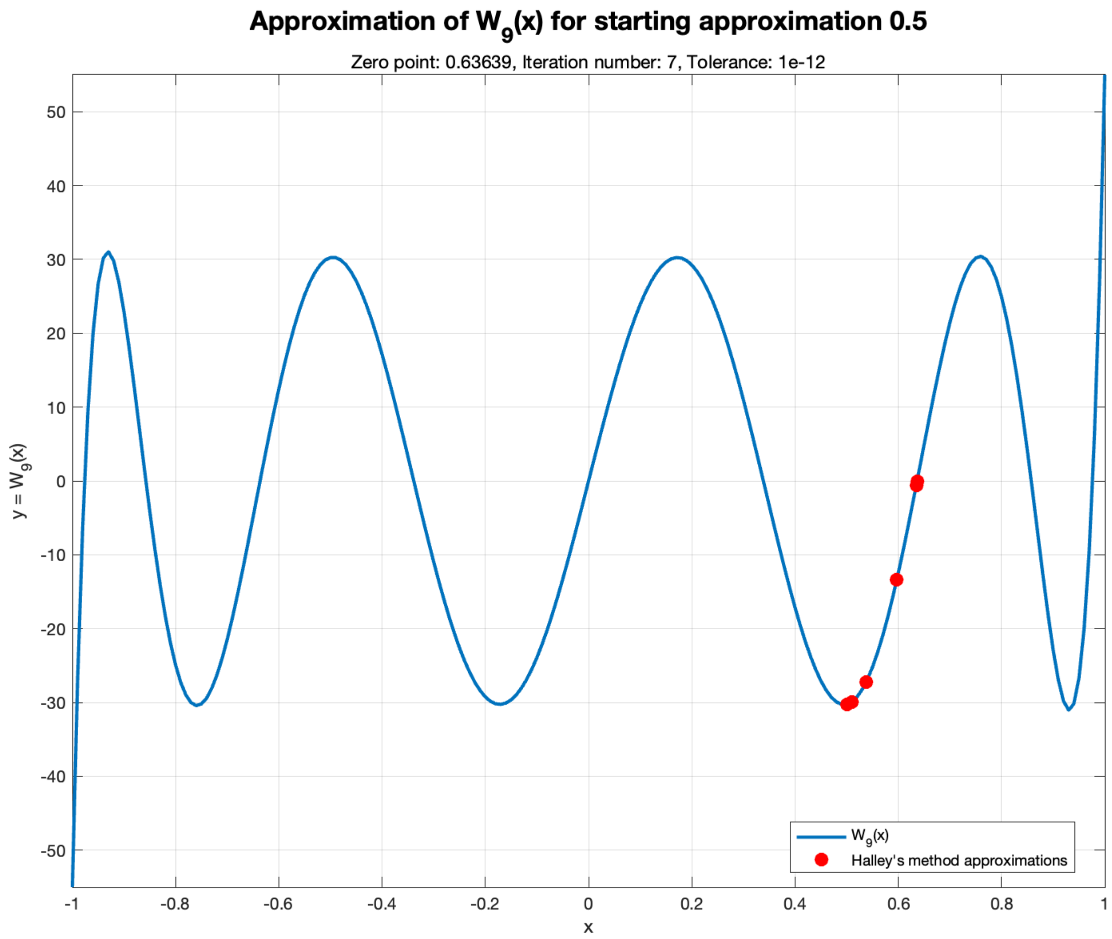
```
% example 1

% define an a sequence
a = 1:10;
fprintf("\na=[" + num2str(a) + "]\n");

% Create W_n of one parameter x
W_func = create_W_func(a);

% plot W_n on range <-1, 1>, and sequence x = (x_1, ...) of next
% approximations of Haley's method for starting approximation of 0.5
% tolerance 10^-12 and maximum iteration number 10^3
plotHalley(W_func, a, 0.5, -1, 1, 0.01, 10^-12, 10^3, "./../plots/example1");

% retrieve x sequence for the same function parameters
z = halley(W_func, 0.5, 10^-12, 10^3);
% print final zero point
fprintf("Final zero point: %.20f \n", z(length(z)));

% calculate error
expected = fzero(W_func, 0.5);
error = calculate_error(expected, z(length(z)));
fprintf("Relative error: " + error + ". Expected zero point is %.20f\n", expecte

% test if method in fact approched it with correct tolerance
assert(abs(error) < 10^-12);
```

Which outputs:

```
a=[1    2    3    4    5    6    7    8    9   10]
Final zero point: 0.63639349519183574522
Relative error: 0. Expected zero point is 0.63639349519183574522
```

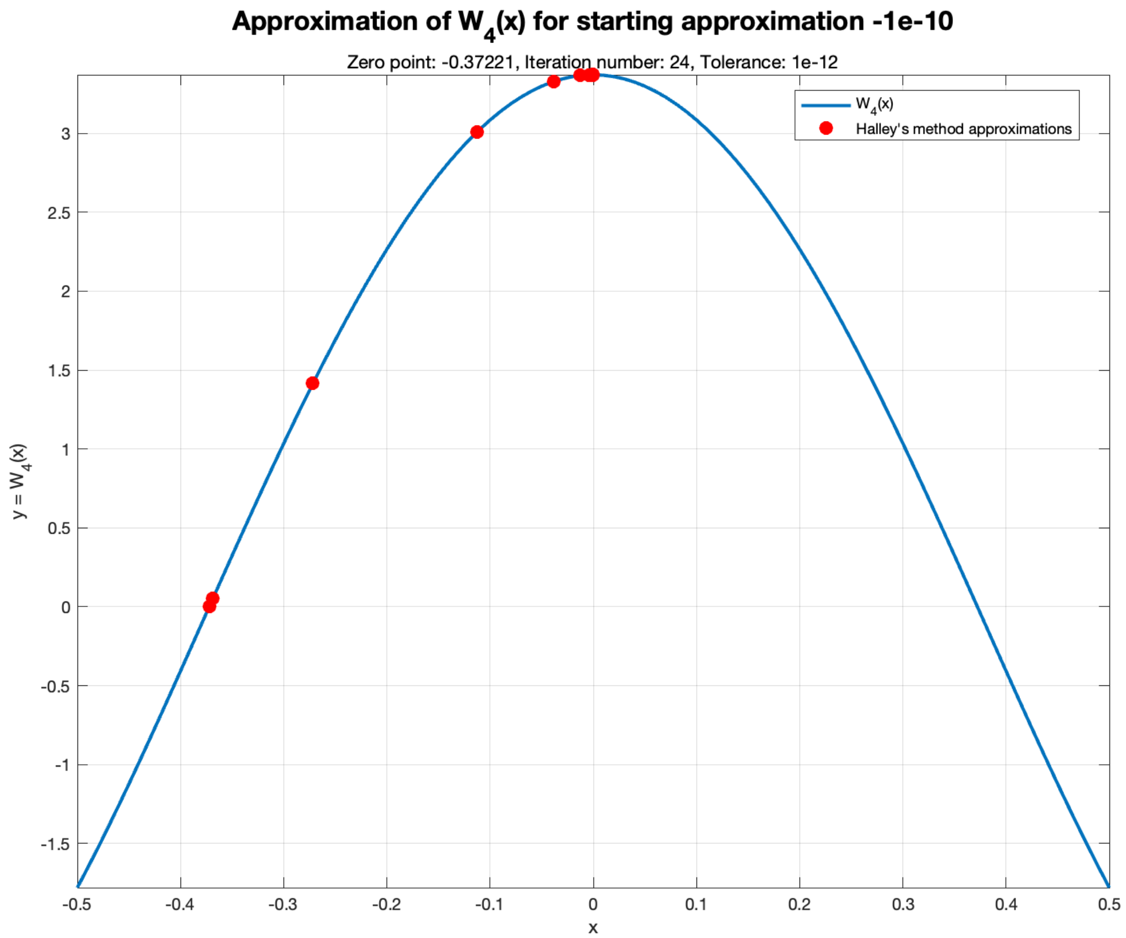
4. Computation examples and analysis

Lets begin with some visual examples:



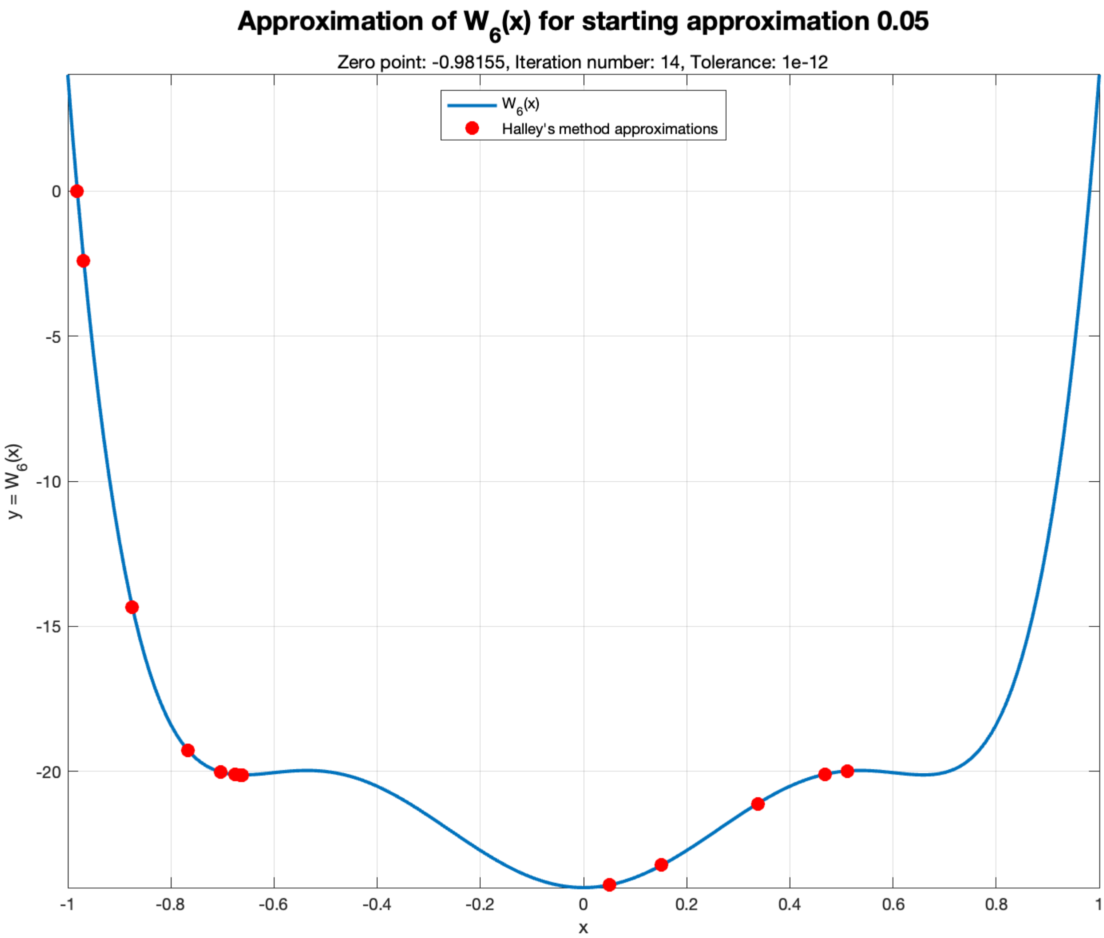
Example 1. Here we begin a little bit to right from the middle of the curve. Algorithm decides to approach towards right and manages to coverage in 7 iterations.

a='1 2 3 4 5 6 7 8 9 10'



Example 2. Here we begin a little bit to left from the center. Note that for center this method will not converage, it'll get stuck there.

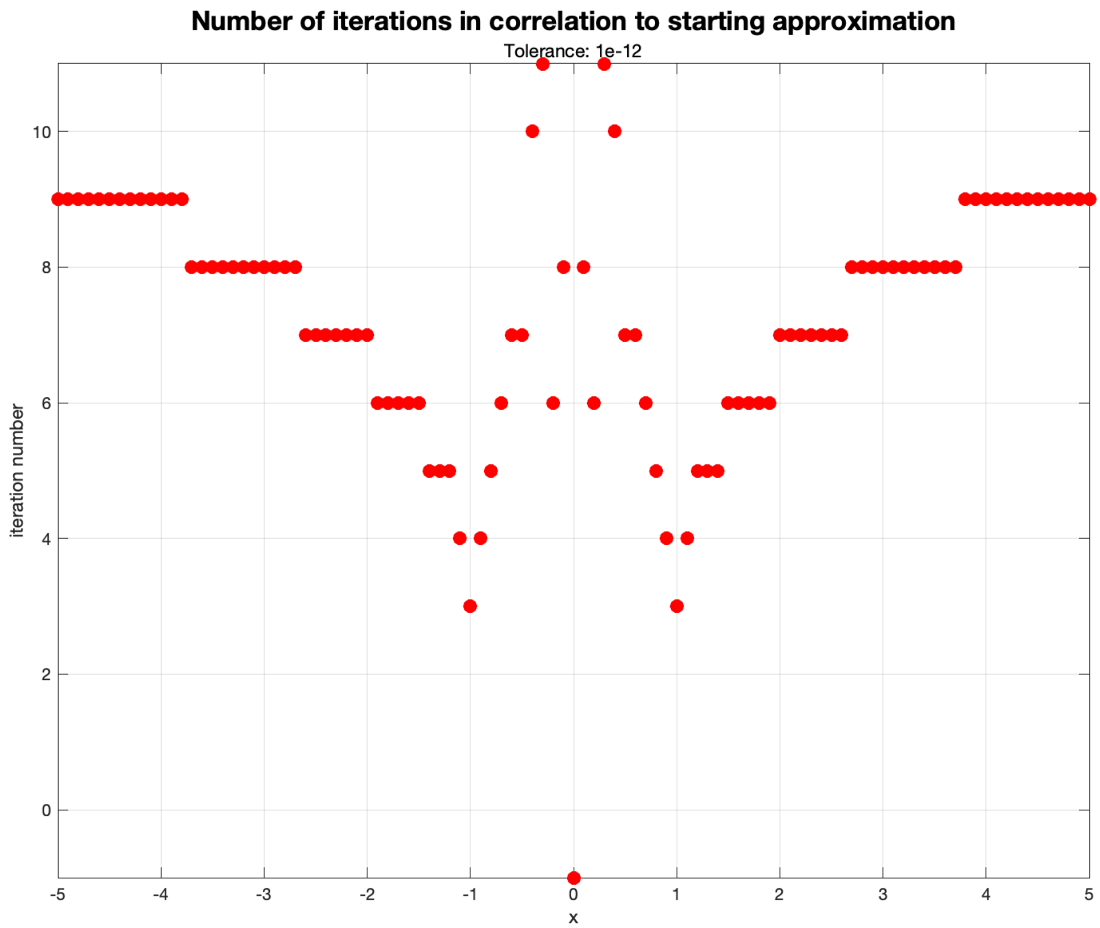
a='-0.45055 1.9936 2.8743 2.5057 0.94723'



Example 3. Here we begin at 0.05, and algorithm scans both to right and the left before reahing coverage at around -1 (interestingly it's further away than coverage point around 0.95).

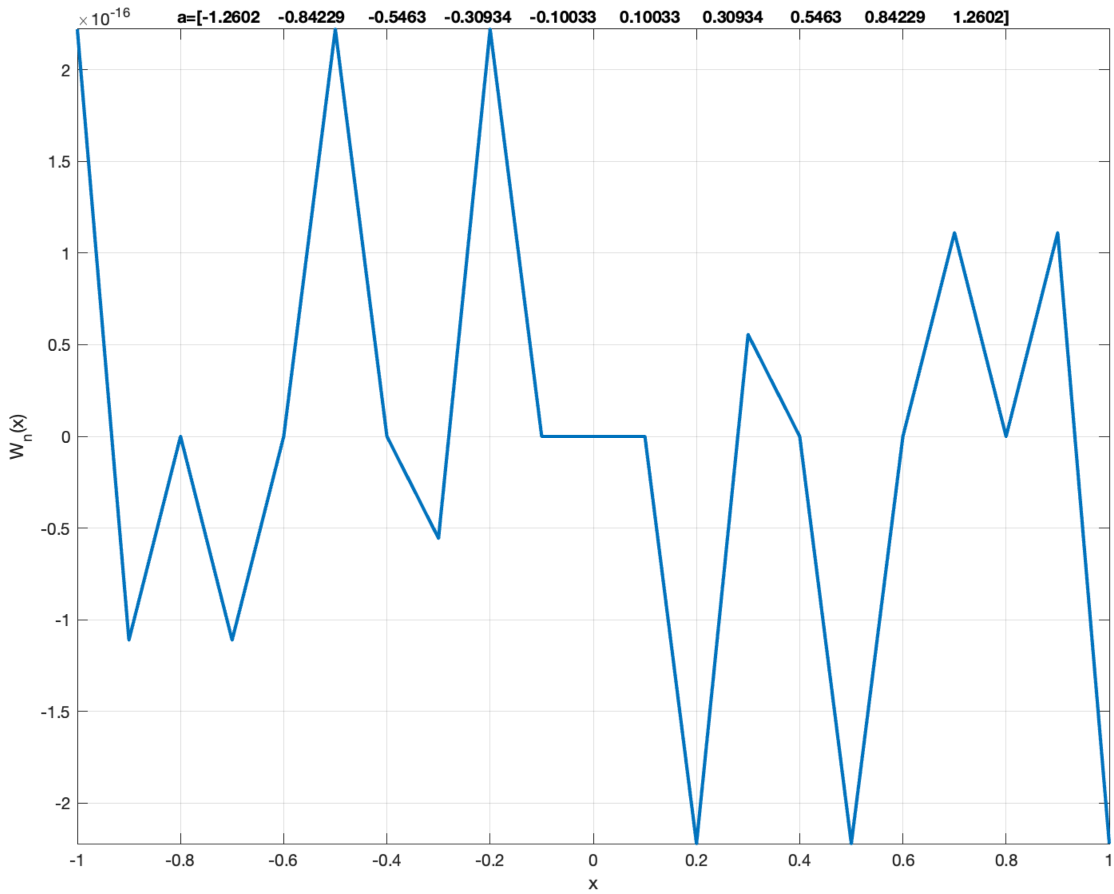
a='-0.45055 1.9936 2.8743 2.5057 0.94723'

Let's now try to analyze how many iterations this method require for such an easy example as the last upper one.

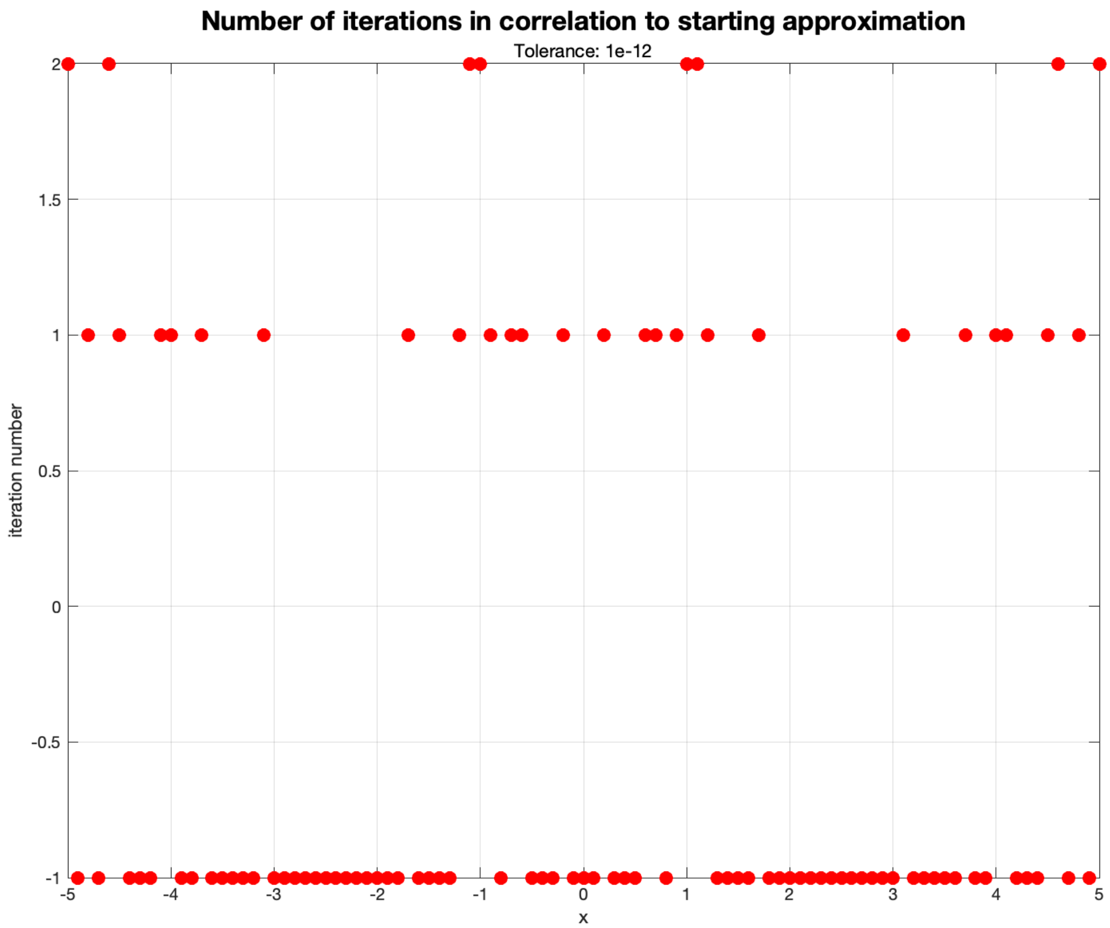


Number of iterations equals to -1 represents that for given maximum iterations (10^3) Hallye's method didn't manage to find the zero point

From above image we can easily tell, which of course checks with the math, that when function derevative is 0 Halley's algorithm gets trapped forever, therefore it exceedes any possible iteration limit and we cannot use it to approximate the zero points. For clarity let's look at this again:

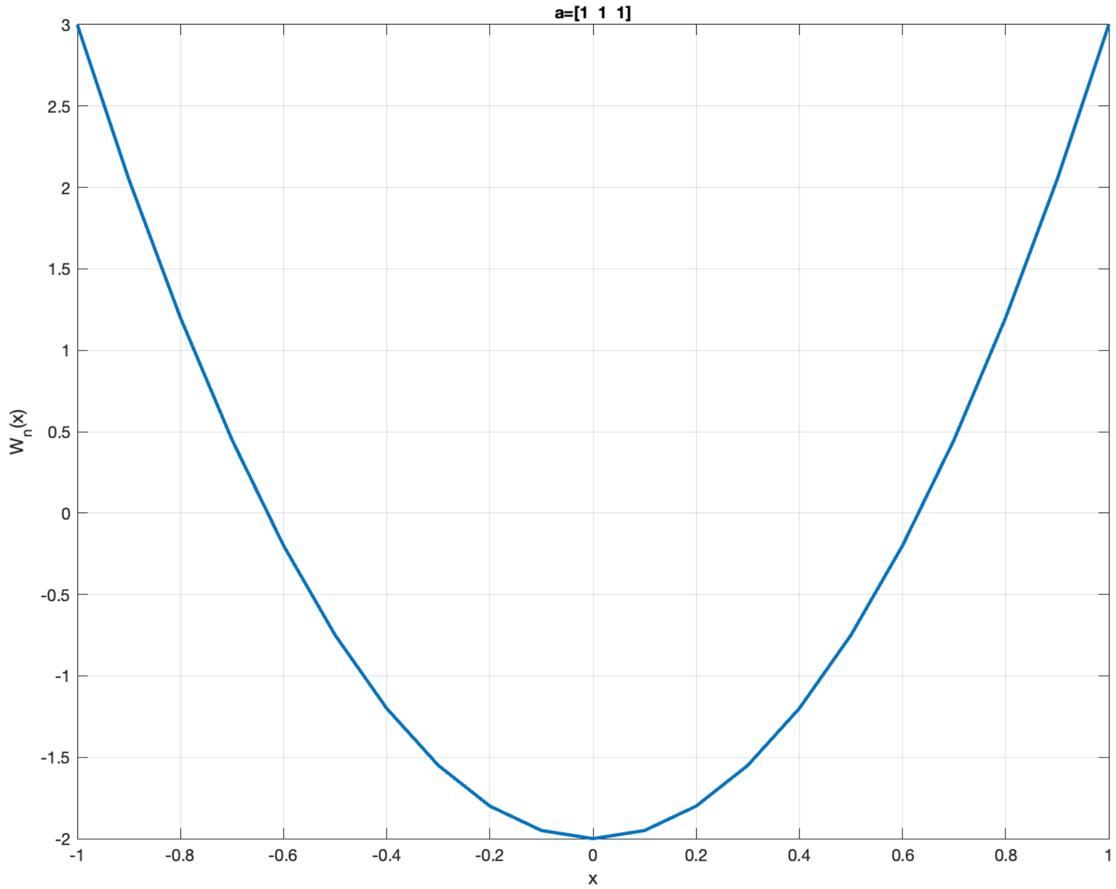


Example 4. Function we'll try to approximate with Halley's method. It has many points where its derevative is equal to 0 and many points where its value is close to 0.

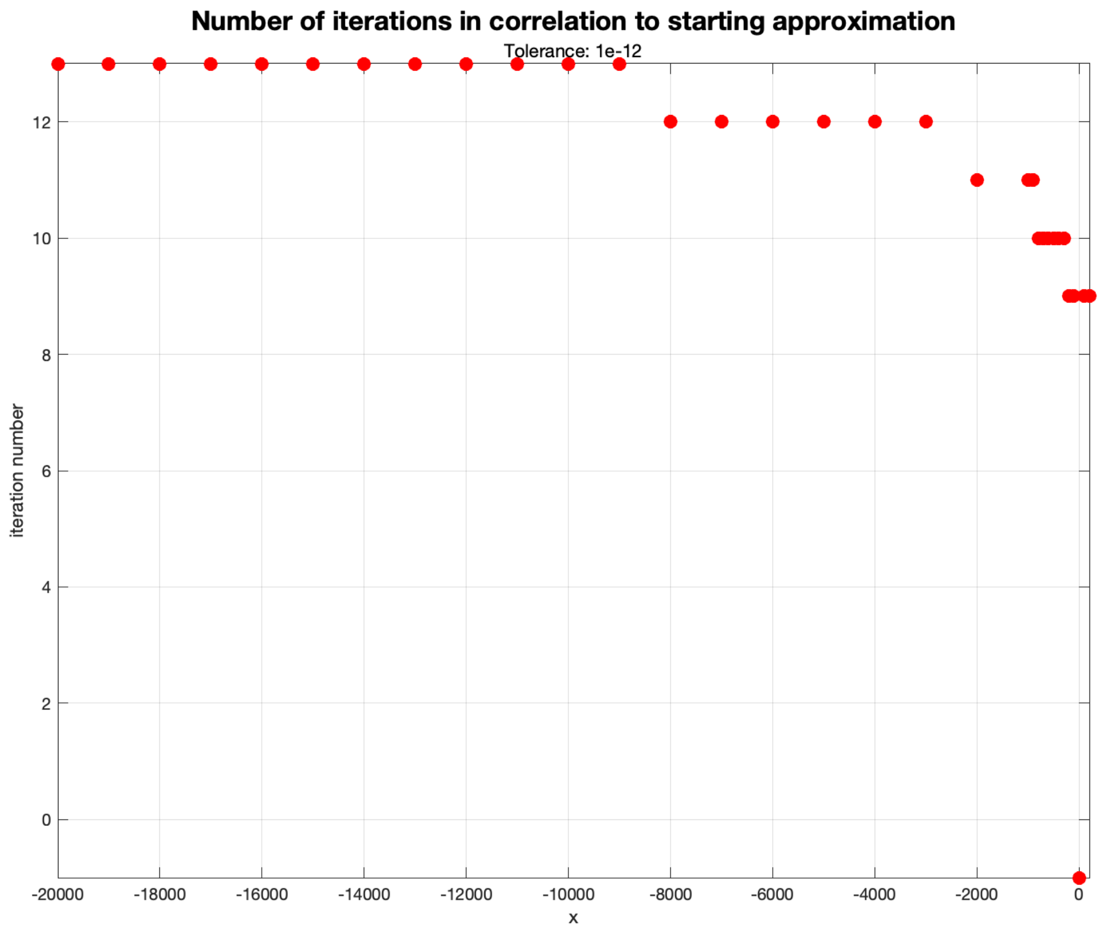


As we've expected, Halley's method either finds optimal solution in couple of iterations or gets stuck.

The only remaining question is how fast can it approach (if it is possible) the desired optimum while being relatively far away from it. To anwser this question let’s look at next 2 charts that tackle this problem using very simple function:

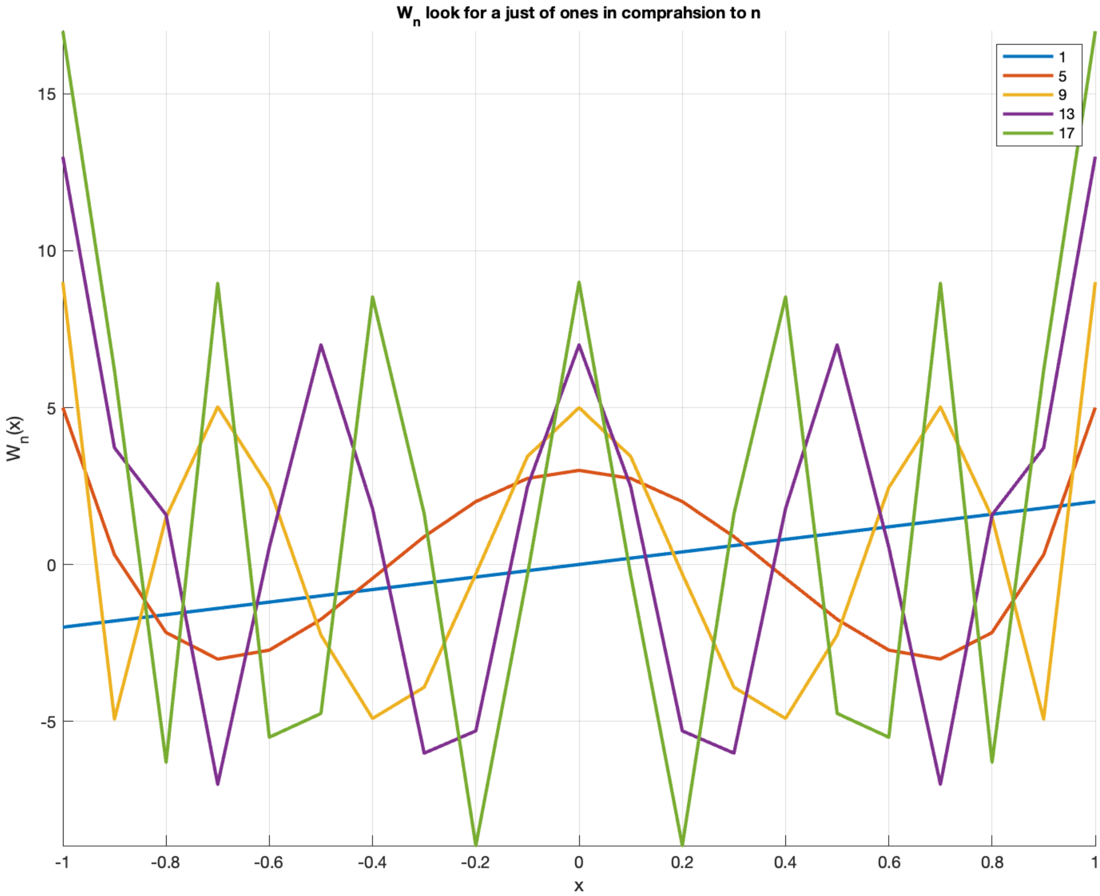


Example 5. Function graphical representation for small x. As we can see, it raises pretty quickly yet it's extreamly simple and smoth.



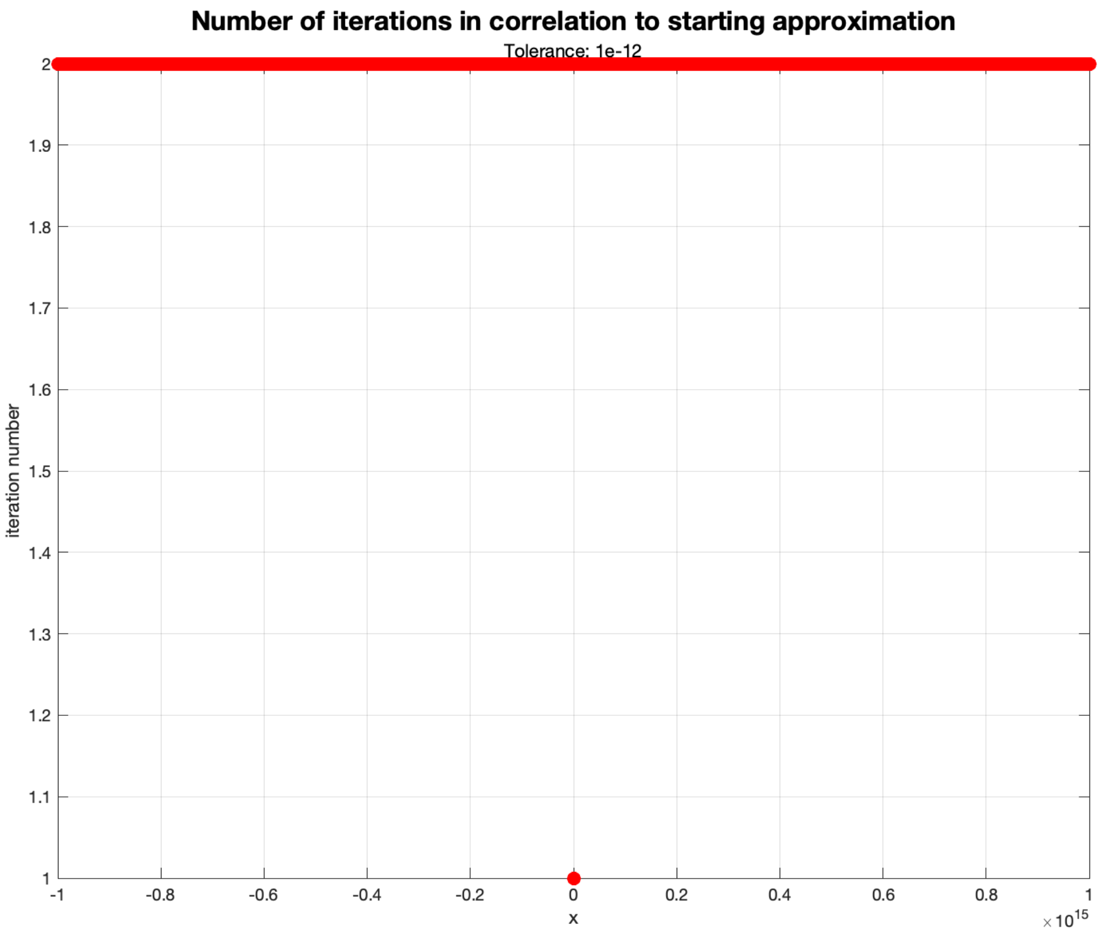
We can se that the higher the function value is - thus in our case the further we are from zero point - the longer it takes to reach it. Note however, that the iterations gain is slow, loking alike logarithmic / square root function.

At the end let's look how does the W_n function looks when $a = (1, 1, \ldots, 1)$ in comprahsion to $n \in \mathbb{N}$:



Example 6. As we can see higher n leads to more bumps in area of 0, however regardless to it's value (except 1) all of these functions begin rappidly growing around -1 and 1

Using what've found out for $a = (1)$, let's look for number of iterations for Halley's method for very large values:



It's nearly constant

Please note that $W_1(10^{15}) = 2000000000000000$ and final zero point for 10^{15} is 0.000000000000000000000, so it's far away - but it's not shocking for us, scince it's linear function so after first iteration we'll have s a point $x_k \in \mathbb{R}$ which is a zero point of a hyperbole that approximates W_n at point x_k, and from linearility of W_n it's exactly equivalent to 0 regardless of how far away we are from that point.

5. Accuracy analysis

Let's now spend a little more time and look what are the accual errors for zero points found for first three examples. We have

Example num	Wolfram zero point	Halley's zerro point (tol=10^-12)	error
1	0.636393495191836	0.63639349519183574522	-3.4891e-16

Example num	Wolfram zero point	Halley’s zerro point (tol=10^-12)	error
2	-0.372210564161923	-0.37221056416192388472	2.3862e-15
3	0.981548363136206	0.98154836313620497101	-1.018e-15

Although relatively small Wolfram accuracy all of the errors are lower than set tolerance of 10^{-12}.

Please note that all of upper examples zero points are checked with zero points returned by `fzero` function from Matlab, however such testing for our implementation was immposible to be automated thus `fzero` can return another zero points even if provided with the same starting approximation.

However, for examples 1-3 and 6 both Wolfram and `fzero` returns zero points that are enough close to our custom Halley’s method zero point to meet set tolerance, which concludes correctness of our implementation based on this 4 test cases.