IMPLEMENTATION OF CHEBYSHEV-HALLEY TYPE METHODS BASED ON HYPER-DUAL NUMBERS

V. I. Olifer

Finding the zeros of non-linear functions quickly and accurately is an interesting, complex, and practically important problem in the field of computational mathematics. This problem is reduced to solving the equation $f(x) = 0$, where $f(x)$ is a scalar non-linear function defined on a certain interval of real numbers. One of the most famous and basic tools for solving such equations is Newton's method, given by the formula:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \qquad i = 0, 1, 2, \ldots \tag{1}$$

It converges quadratically for simple roots and linearly for multiple roots. In the development of this method, cubically convergent one-point methods were proposed. One of such well-known schemes is the classical cubically convergent Chebyshev-Halley family, which improves the Newton method and is given by the following relation [1]:

$$x_{i+1} = x_i - \left[1 + \frac{1}{2} \cdot \frac{L(x_i)}{1 - \alpha L(x_i)}\right] D(x_i), \qquad \alpha \in R \tag{2}$$

where: $L(x_i) = \frac{f(x_i) f''(x_i)}{f'(x_i)^2} = D(x_i) \cdot f''(x_i)/f'(x_i)$ and $D(x_i) = f(x_i)/f'(x_i)$ is the Newtonian correction.

Different values of α in (2) give different (known and unknown) iterative formulas.

For $\alpha = 0$ we have the formula of the Chebyshev method:

$$x_{i+1} = x_i - \left[1 + \frac{1}{2} \cdot L(x_i)\right] D(x_i), \tag{3}$$

and for $= 1/2$ we get the formula of the Halley method:

$$x_{i+1} = x_i - \left[1 + \frac{L(x_i)}{2 - L(x_i)}\right] D(x_i), \tag{4}$$

and, finally, for $\alpha = 1$ we have the super-method Halley formula:

$$x_{i+1} = x_i - \left[1 + \frac{1}{2} \cdot \frac{L(x_i)}{1 - L(x_i)}\right] D(x_i) \tag{5}$$

This family of iterative methods, as was said, has cubic convergence and is associated with the need to calculate $f(x_i)$, $f'(x_i)$ and $f''(x_i)$. In some cases, the calculation of $f'(x_i)$ and especially $f''(x_i)$ is very resource-intensive, and sometimes impossible. Therefore, the direct use of formula (2) is of very limited. To eliminate the second order derivative from $L(x_i)$, approaches based on the approximation of $f''(x_i)$ through $f(x_i)$ and $f'(x_i)$ can be used, i.e. $f''(x_i) \approx \varphi(f(x_i), f'(x_i))$ (see, for example, [1 – 4]). In turn, the calculation of $f'(x_i)$ can also be labor-intensive. And if the first derivative $f'(x_i)$ is also approximated i.e. $f'(x_i) \approx \psi(f(x_i))$, then $f''(x_i) \approx \varphi(f(x_i), \psi(f(x_i)))$, which leads to the imposition of the error $f'(x_i)$ by $f''(\ \ _i)$ and lowering the order of convergence.
Obviously, whatever the approximation model is, it will always contain an error of the model itself. However, the problem of accuracy of computer calculation of derivatives of a function can be successfully solved by using the method of automatic differentiation (AD), the essence of which is to create a new data type and redefine operations on them. This approach allows one to determine exact (with machine precision) values of a function and its derivatives. For the case under consideration, truncated hyper-dual numbers can be used as such a new data type [5].
According to [5], a truncated hyper-dual number is defined by the expression $X = x + x_1\boldsymbol{\varepsilon} + x_2\boldsymbol{\omega}$, where x, x_1 and x_2 are real numbers, $\boldsymbol{\varepsilon}$ and $\boldsymbol{\omega}$ are imaginary symbols. The space of truncated hyper-dual numbers corresponds to a three-dimensional algebra with the rule of multiplication of basis elements $\{\mathbf{1}, \ \boldsymbol{\varepsilon}, \ \boldsymbol{\omega}\}$:

$\times$	***1***	$\boldsymbol{\varepsilon}$	$\boldsymbol{\omega}$
1	*1*	$\boldsymbol{\varepsilon}$	$\boldsymbol{\omega}$
$\boldsymbol{\varepsilon}$	$\boldsymbol{\varepsilon}$	$2\boldsymbol{\omega}$	*0*
$\boldsymbol{\omega}$	$\boldsymbol{\omega}$	*0*	*0*

(6)

Table 1. Rules for multiplying elements of the basis of truncated hyper-dual numbers

The number $x = Re(X) = X.Re$ is called the main part of X, and $x_1 = Im_1(X) = X.Im1$, $x_2 = Im_2(X) = X.Im2$ - imaginary parts of X.
The algebraic operations of addition, multiplication, inversion and division (taking into account *Table 6*) are defined according to the rules:

$$A = a + a_1\boldsymbol{\varepsilon} + a_2\boldsymbol{\omega}, \ B = b + b_1\boldsymbol{\varepsilon} + b_2\boldsymbol{\omega},$$
$$A + B = a + b + (a_1 + b_1)\boldsymbol{\varepsilon} + (a_2 + b_2)\boldsymbol{\omega},$$
$$A{\cdot}B = a{\cdot}b + (a{\cdot}b_1 + b{\cdot}a_1)\boldsymbol{\varepsilon} + (a{\cdot}b_2 + 2{\cdot}a_1{\cdot}b_1 + b{\cdot}a_2)\boldsymbol{\omega},$$
$$A^{-1} = a^{-1} - a_1{\cdot}a^{-2}\boldsymbol{\varepsilon} + (2{\cdot}a_1^2{\cdot}a^{-3} - a_2{\cdot}a^{-2})\boldsymbol{\omega}, \qquad (7)$$
$$\mathrm{A}/_{B} = \mathrm{A}{\cdot}\mathrm{B}^{-1} = a{\cdot}b^{-1} + (a_1{\cdot}b^{-1} - a{\cdot}b_1{\cdot}b^{-2})\boldsymbol{\varepsilon} + [2{\cdot}(a{\cdot}b_1^2{\cdot}b^{-3} - a_1{\cdot}b_1{\cdot}b^{-2}) - a{\cdot}b_2{\cdot}b^{-2} + a_2{\cdot}b^{-1}]\boldsymbol{\omega}$$

The truncated hyper-dual argument function is implemented by the expression:

$$F(X) = f(x) + x_1 \cdot f'(x)\boldsymbol{\varepsilon} + \ (x_2 \cdot f'(x) + x_1^2 \cdot f''(x)\ )\boldsymbol{\omega} \tag{8}$$

For $x_1 = 1$ and $x_2 = 0$, formula (8) takes the form:

$$F(X) = f(x) + f'(x)\boldsymbol{\varepsilon} \ + \ f''(x)\boldsymbol{\omega} \tag{9}$$

The description of elementary (basic) functions of a truncated hyper-dual argument is given in [5]. For example, $\ln(X) = \ln(x) + x^{-1}\boldsymbol{\varepsilon} - x^{-2}\boldsymbol{\omega}$, whence $Re(X) = f(x) = \ln(x)$, $Im_1(X) = f'(x) = x^{-1}$ and $Im_2(X) = f''(x) = -x^{-2}$.

The calculation of a complex truncated hyper-dual function (function composition) of the form $F = \ f_1(f_2(\dots f_k(X) \dots), X), X)$ (where the value of f_k is used as an argument for f_{k-1}) must be started with calculations $F_k = \ f_k(X)$, continue with calculations $F_{k-1} = \ f_{k-1}(F_k, X)$, $F_{k-2} = f_{k-2}(F_{k-1}, X), \dots, F = \ f_1(F_2, X)$.

To apply AD (based on truncated hyper-dual numbers and truncated hyper-dual functions) to relations (2), it is necessary to apply the following mappings

$$x \ \rightarrow \ X = x + 1 \cdot \boldsymbol{\varepsilon} + \ 0 \cdot \boldsymbol{\omega}, \quad f(x) \rightarrow F(X) = f(x) + f'(x) \cdot \boldsymbol{\varepsilon} + \ f''(x) \cdot \boldsymbol{\omega}$$

Then for (2) we have

$$D(x_i) = \ F(X_i).Re/\,F(X_i).Im_1, \quad L(x_i) = D(x_i)[F(X_i).Im_1/F(X_i).Im_2], \tag{10}$$

where: $X_i = x_i + 1 \cdot \boldsymbol{\varepsilon} + \ 0 \cdot \boldsymbol{\omega}$

Using the SWIFT code describing the Thdn (truncated hyper-dual number) data type [5], a computer implementation of the above approach was performed in the SWIFT 5 programming language for macOS 13.3. The procedure used in the numerical experiments ChebyshevHalley(.) (see Appendix 1) implements the iterative formula (2) taking into account (10). The procedure ChebyshevHalleyX(.) is also given, taking into account the area limiting the search for a solution.

To carry out a numerical analysis of the proposed approach, a wide variety of initial functions were considered. Some calculation results are shown in *Table 2*.

$f(x)$	x_0	$\boldsymbol{n}$ depending on the value α				
		$\alpha = 0$	$\alpha = 1/4$	$\alpha = 1/2$	$\alpha = 3/4$	$\alpha = 1$
$x^3 + 4x^2 - 10$	1.0	5	4	4	4	4
$\cos(x) - x$	0.0	5	5	5	4	4
$(x-1)^3 - 1$	1.5	5	5	5	5	5
$x^3 + sin^2(x) + 3\cos(x) + 5$	-2.0	4	4	4	4	4

$e^{-x} + \cos(x)$	1.0	4	4	4	4	4
$x^2 - e^x - 3x + 2$	0.0	4	4	4	4	4
$e^{x^2+7x-30} - 1$	3.1	5	5	5	4	5
$\sin(x)$	-0.6	6	5	4	4	4

*Table 2. Comparison of the number of iterations **n** depending on the value of α.*

The obtained numerical results fully correspond to the numerical solutions given in [3], where the procedures for approximating $f''(x)$ based on the arithmetic, counterharmonic and centroidal average values of $f(x)$ and $f'(x)$ were used, which leads to overly grammatical formulas related to fourth-order two-point methods, for example,

$$x_{i+1} = x_i - \\ - D(x_i)\left[\frac{3f(x_i)^3 - 12f(x_i)f(y_i)^2(\alpha-1) - 16f(y_i)^3(\alpha-1)^2(2\alpha-1) + 3f(x_i)^2f(y_i)(4\alpha-5)}{3f(y_i)^3 - 24f(x_i)f(y_i)^2(\alpha-1) - 32f(y_i)^3(\alpha-1)^2\alpha + 6f(x_i)^2f(y_i)(2\alpha-3)}\right],$$

where: $y_i = x_i - D(x_i) = x_i - f(x_i)/f'(x_i)$ and $f(x), f'(x)$ must be given by analytic expressions.
On the other hand, the variant based on automatic differentiation using special dual numbers (truncated hyper-dual numbers) is actually implemented by relations (2), (10) and the new data type Thdn. This requires an analytical representation of only the function under study, which can be any composition of basic functions and is determined by the program code.

Appendix 1.

Code for a numerical experiment in Swift 5 (macOS 13.3). The data type Thdn (truncated-dual number) is given in [5], or you can download static library ThdnLibraryX.zip and after unzip add ThdnLibraryX into your project (how to do it read ReadMe.txt). Download ThdnLibraryX.zip

```
// To implement the code below, you need to add a file that describes the data type Thdn (см. [5])
//or static library ThdnLibraryX

import Foundation;

import ThdnLibraryX;    // add this one if you use static library ThdnLibraryX

let δ:Double = 1E-15;    // allowable error

// INPUT DATA:
// f – function pointer, xo – start point, α –  method constant
// OUTPUT DATA:
// (x, i) – solution and number of iterations
```

```
func ChebyshevHalley(f:(Double) -> Thdn, xo:Double, α:Double)->(x:Double, i:Int){
   var xi = xo, x = 0.0, E = Double.greatestFiniteMagnitude;
   var L, D: Double, i = 0, F = Thdn();
   while(E >= δ){
      F = f(xi);  D = F.re/F.im1;   L = D*F.im2/F.im1;
      x = xi - (1.0 + 0.5*L/(1.0 - α*L))*D;
      E = abs(x - xi);   xi = x;   i += 1;
   }
   return (x, i);
}

// INPUT DATA:
// f – function pointer, xo – start point, α –  method constant, ab() solution search area boundaries
// OUTPUT DATA:
// (x, i) – solution and number of iterations

func ChebyshevHalleyX(f:(Double) -> Thdn,x0:Double, α:Double,
                    ab:(a:Double, b:Double))->(x:Double,i:Int){
   var xi = x0, x = 0.0, E = Double.greatestFiniteMagnitude;
   var L, D: Double, i = 0, F = Thdn();
   while(E >= δ){
      F = f(xi);  D = F.re/F.im1;   L = D*F.im2/F.im1;
      x = xi - (1.0 + 0.5*L/(1.0 - α*L))*D;
      if x > ab.b {x = ab.b}  else if  x < ab.a {x = ab.a}
      E = abs(x - xi);   xi = x;   i += 1;
   }
   return (x, i);
}

// function calls ChebyshevHalley () and = ChebyshevHalleyX ()

 let r  = ChebyshevHalley(f: func7, x0: 2.8, α: 0.5);
 let rr = ChebyshevHalleyX(f: func7, x0: 2.8, α: 0.5,ab: (2.8, 3.5));

//  functions under study

func func1(x:Double)->Thdn{
   let X = Thdn(re: x, im1: 1, im2: 0);
   return   X**3 + 4.0*X**2 - Thdn(re: 10.0);
}

func func2(x:Double)->Thdn{
   let X = Thdn(re: x, im1: 1, im2: 0);
   return   Thdn.cos(X: X) – X;
}

func func3(x:Double)->Thdn{
   let X = Thdn(re: x, im1: 1, im2: 0);
   let Z = X - Thdn(re: 1.0)
   return   Z**3 - Thdn(re: 1.0);
}

func func4(x:Double)->Thdn{
   let X = Thdn(re: x, im1: 1, im2: 0);
   return   X**3 + Thdn.sin(X: X)**2 + 3.0*Thdn.cos(X: X) + Thdn(re: 5.0);
}
```

```
func func5(x:Double)->Thdn{
   let X = Thdn(re: x, im1: 1, im2: 0);
   return   Thdn.exp(X: -X) + Thdn.cos(X: X);
}

func func6(x:Double)->Thdn{
   let X = Thdn(re: x, im1: 1, im2: 0);
   return   X**2 - Thdn.exp(X: X) - 3.0*X + Thdn(re: 2.0);
}

func func7(x:Double)->Thdn{
   let X = Thdn(re: x, im1: 1, im2: 0);
   let Z = X**2 + Thdn(re: 7.0)*X - Thdn(re: 30.0)
   return   Thdn.exp(X: Z) - Thdn(re: 1.0);
}

func func8(x:Double)->Thdn{
   let X = Thdn(re: x, im1: 1, im2: 0);
   return   Thdn.sin(X: X);
}
```

REFERENCES

1. *Argyros I. K., Kansal M., Kanwar V., Bajaj S.* Higher-order derivative-free families of Chebyshev-Halley type methods with or without memory for solving nonlinear equations, Appl. Math. Comput. 315 (2017), 224– 245.

2. *Zhanlav T., Chuluunbaatar O.* On some iterative methods of high order of convergence for solving nonlinear equations. - //Bulletin of RUDN University / Mathematician series. Computer science. Physics, 2009. - No. 4. - p. 47-55.

3. *Kansal M., Kanwar V., Bhatia S.* Optimal properties of methods of the Chebyshev-Halley type without a second derivative based on average values.- //Comput. Mathematics/RAS Sib. department. - Novosibirsk, 2016. - V.19, No. 2. - pp.167-181.

4. *Gander V.* New algorithms for solving the nonlinear eigenvalue problem. - Comput. mathematics and mat. Physics. - 2021. V.61, No. 5. - p.787-799.

5. *Olifer V. I.* Truncated hyper-dual numbers in automatic differentiation. – URL: http://viosolutions.amerihomesrealty.com/pdf/
Усеченные_гипер-дуальные_числа_в_автоматическом_дифференцировании.pdf, 2020.

Abstract

This paper considers a method for implementing iterative Chebyshev-Halley formulas based on automatic differentiation using special dual numbers (truncated hyper-dual numbers). A computer implementation of this approach for the SWIFT language of the macOS operating system is presented. Numerical experiments have been carried out.

Keywords: *iterative methods for solving nonlinear equations, automatic differentiation, truncated hyper-dual numbers.*

August 20, 2023