РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ НА ОСНОВЕ АВТОМАТИЧЕСКОГО ДИФФЕРЕНЦИРОВАНИЯ И УСЕЧЕННЫХ ГИПЕР-ДУАЛЬНЫХ ЧИСЕЛ

Олифер В.И.

1. Автоматическое дифференцирование

Точность вычислений является основным критерием качества результатов вычислений. Любой вычислительный процесс содержит погрешность метода или усечения и погрешность конечного округления. Так, например, вычисление базовой функции e^x осуществляется путем ее разложения в ряд Тейлора с остаточным членом $\mathcal{O}(k)$

$$e^x = \sum_{n=0}^{\infty} x^n/n! = \sum_{n=0}^{k} x^n/n! + \sum_{n=k+1}^{\infty} x^n/n! = \sum_{n=0}^{k} x^n/n! + \mathcal{O}(k),$$

где остаточный член $\mathcal{O}(k)$ и есть погрешность метода. Величина k определяется из условия: $\mathcal{O}(k)$ должен быть меньше точности представления данных.

Погрешность округления связана с точностью представления данных (количеством значащих цифр после запятой). Для данных с плавающей точкой компьютер выделяет определенный размет дробной части числа, не вмещающиеся цифры отбрасываются без округления. Например, пусть размер дробной части равен 15-ти цифрам и $x = 0.000000000000001$, а $y = 0.1$, тогда $x{\cdot}y = 0.000000000000000$, что приводит к потери результата.

Традиционные способы вычисления производной $f'(x)$ на практике сопряжены с трудностями и имеют существенные недостатки. Реализация вычислений $f'(x)$ в виде отдельной программной процедуры является избыточным решением и требует предварительного анализа $f(x)$. Использование метода конечных разностей нуждается в компромиссе между погрешностями усечения и округления, т. е. нахождения оптимальной величины приращения аргумента x. Более того, задача численного дифференцирования является некорректной и ее нужно решать методами регуляции [1]. Еще большие проблемы возникают при вычислении второй производной $f''(x)$.

Обойти указанные трудности позволяет метод автоматического дифференцирования (AD), связанный с вычислением точных производных функции, представленных компьютерным кодом [2]. Этот метод является фундаментальным инструментом в задачах, связанных с оптимизаций, нелинейными уравнениями, дифференциальными уравнениями, анализе чувствительности и т. д.

Для компьютерной реализации AD необходимо создать новый тип данных, перезагрузить базовые функции и операции над ними. Если новый тип данных строится на основе обычных дуальных чисел [3], то за одно обращение к перезагруженной базовой функции точно (вплоть до погрешности округления) вычисляются значение самой функции и ее производной. Этого вполне достаточно чтобы реализовать вычислительный процесс, не использующей

производные выше первого порядка (например, классический метод Ньютона [4]). Если же вычислительный процесс требует использование производных выше первого порядка (например, метод Ньютона-Чебышева [4]), то необходимо расширение дуальных чисел, которые называются гипер-дуальными числами [5, 6].

2. Гипер-дуальные числа

Пространство гипер-дуальных чисел отвечает четырехмерной алгебре с правилом умножения элементов базиса $\{\mathbf{1},\ \boldsymbol{\varepsilon}_1,\ \boldsymbol{\varepsilon}_2,\ \boldsymbol{\varepsilon}_{12}\}$:

$\times$	$\mathbf{1}$	$\boldsymbol{\varepsilon}_1$	$\boldsymbol{\varepsilon}_2$	$\boldsymbol{\varepsilon}_{12}$
$\mathbf{1}$	1	$\boldsymbol{\varepsilon}_1$	$\boldsymbol{\varepsilon}_2$	$\boldsymbol{\varepsilon}_{12}$
$\boldsymbol{\varepsilon}_1$	$\boldsymbol{\varepsilon}_1$	0	$\boldsymbol{\varepsilon}_{12}$	0
$\boldsymbol{\varepsilon}_2$	$\boldsymbol{\varepsilon}_2$	$\boldsymbol{\varepsilon}_{12}$	0	0
$\boldsymbol{\varepsilon}_{12}$	$\boldsymbol{\varepsilon}_{12}$	0	0	0

(2.1)

Табл. 1. Правила умножения элементов базиса гипер-дуальных чисел

При этом гипер-дуальное число представимо в виде:

$$\mathcal{X} = x + x_1\boldsymbol{\varepsilon}_1 + x_2\boldsymbol{\varepsilon}_2 + x_3\boldsymbol{\varepsilon}_{12}\ , \tag{2.2}$$

а гипер-дуальная функция определяется выражением:

$$F(\mathcal{X}) = f(x) + x_1{\cdot}f'(x)\boldsymbol{\varepsilon}_1 + \ x_2{\cdot}f'(x)\boldsymbol{\varepsilon}_2 + \ (x_3{\cdot}f'(x) + x_1{\cdot}x_2{\cdot}f''(x))\boldsymbol{\varepsilon}_{12} \tag{2.3}$$

В соотношениях (2.1) ÷ (2.3): $\boldsymbol{\varepsilon}_1$, $\boldsymbol{\varepsilon}_2$ и $\boldsymbol{\varepsilon}_{12}$ – мнимые символы, x, x_1, x_2 и x_3 – вещественные числа, а $f'(x)$ и $f''(x)$ – первая и вторая производные функции $f(x)$ соответственно. Число $x = Re(\mathcal{X})$ называется главной частью $\mathcal{X}$, а $x_1 = Im_1(\mathcal{X})$, $x_2 = Im_2(\mathcal{X})$, $x_3 = Im_3(\mathcal{X})$ мнимыми частями $\mathcal{X}$.

Заметим, что при $x_1 = x_2 = 1$ и $x_3 = 0$ соотношение (3) упрощается и принимает вид:

$$F(\mathcal{X}) = f(x) + f'(x)\boldsymbol{\varepsilon}_1 + f'(x)\boldsymbol{\varepsilon}_2 \ + \ f''(x)\boldsymbol{\varepsilon}_{12} \tag{2.4}$$

Для итерационных процессов, использующих значения функции, ее первой и второй производных, выражение (2.4) является избыточным т. к. первая и вторая мнимые части содержат одну и ту же величину – значение первой производной.

3. Усечённые гипер-дуальные числа

Чтобы избавиться от избыточности гипер-дуальных функций, указанной в предыдущем параграфе, введем базис $\{\mathbf{1},\ \boldsymbol{\varepsilon},\ \boldsymbol{\omega}\}$ с правилом умножения:

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

(3.1)

Табл. 2. Правила умножения элементов базиса усечённых гипер-дуальных чисел

Новое гипер-дуальное число запишется так $X = x + x_1\boldsymbol{\varepsilon} + x_2\boldsymbol{\omega}$, которое будем называть *усечённым гипер-дуальным числом* (truncated-hyper-dual number). Раскладывая $F(X)$ в ряд Тейлора, получим

$$F(X) = f(x) + x_1 \cdot f'(x)\boldsymbol{\varepsilon} + (x_2 \cdot f'(x) + x_1^2 \cdot f''(x))\boldsymbol{\omega} \tag{3.2}$$

При $x_1 = 1$ и $x_2 = 0$ выражение (3.2) принимает вид:

$$F(X) = f(x) + f'(x)\boldsymbol{\varepsilon} + f''(x)\boldsymbol{\omega} \tag{3.3}$$

Полученное соотношение (3.3) свободно от избыточности, присутствующей в (2.4).
Алгебраические операции сложения, умножения, обращения и деления (с учетом табл. 2) определены по правилам:

$$\begin{aligned}
&A = a + a_1\boldsymbol{\varepsilon} + a_2\boldsymbol{\omega}, \qquad B = b + b_1\boldsymbol{\varepsilon} + b_2\boldsymbol{\omega} \\
&A + B = a + b + (a_1 + b_1)\boldsymbol{\varepsilon} + (a_2 + b_2)\boldsymbol{\omega}, \\
&A \cdot B = a \cdot b + (a \cdot b_1 + b \cdot a_1)\boldsymbol{\varepsilon} + (a \cdot b_2 + 2 \cdot a_1 \cdot b_1 + b \cdot a_2)\boldsymbol{\omega}, \\
&A^{-1} = a^{-1} - a_1 \cdot a^{-2}\boldsymbol{\varepsilon} + (2 \cdot a_1^2 \cdot a^{-3} - a_2 \cdot a^{-2})\boldsymbol{\omega}, \\
&{}^{A}/_{B} = A \cdot B^{-1} = a \cdot b^{-1} + (a_1 \cdot b^{-1} - a \cdot b_1 \cdot b^{-2})\boldsymbol{\varepsilon} + [2 \cdot (a \cdot b_1^2 \cdot b^{-3} - a_1 \cdot b_1 \cdot b^{-2}) - \\
&\qquad\qquad a \cdot b_2 \cdot b^{-2} + a_2 \cdot b^{-1}]\boldsymbol{\omega}
\end{aligned} \tag{3.4}$$

Воспользовавшись [7], приведем компоненты некоторых элементарных (базовых) функций усечённого гипер-дуального аргумента $X = x + x_1 \cdot \boldsymbol{\varepsilon} + x_2 \cdot \boldsymbol{\omega}$:

$F(X)$	$\boldsymbol{f(x)}$	$\boldsymbol{f'(x)}$	$\boldsymbol{f''(x)}$
X^n	x^n	$n \cdot x^{n-1}$	$n \cdot (n-1) \cdot x^{n-2}$
e^X	e^x	e^x	e^x
a^X	a^x	$a^x \ln(a)$	$a^x \cdot \ln(x)^2$
$\ln(X)$	$\ln(x)$	x^{-1}	$-x^{-2}$
$\log_a(X)$	$\log_a(x)$	$(x \ln(a))^{-1}$	$-x^{-2} \cdot \ln(a)^{-1}$
$\sin(X)$	$\sin(x)$	$\cos(x)$	$-\sin(x)$
$\cos(X)$	$\cos(x)$	$-\sin(x)$	$-\cos(x)$
$\mathrm{tg}(X)$	$\mathrm{tg}(x)$	$\sec^2(x) = \cos^{-2}(x)$	$2 \cdot \mathrm{tg}(x) \cdot \cos^{-2}(x)$
$\mathrm{ctg}(X)$	$\mathrm{ctg}(x)$	$-\csc^2(x) = -\sin^{-2}(x)$	$-2 \cdot \mathrm{ctg}(x) \cdot \sin^{-2}(x)$
$\sec(X)$	$\sec(x)$	$\sec(x) \cdot \mathrm{tg}(x)$	$\sec(x) \cdot (2 \cdot \mathrm{tg}^2(x) + 1)$
$\csc(X)$	$\csc(x)$	$-\csc(x) \cdot \mathrm{ctg}(x)$	$\csc(x) \cdot (2 \cdot \mathrm{ctg}^2(x) + 1)$
$\arcsin(X)$	$\arcsin(x)$	$1/\sqrt{1-x^2}$	$x \cdot (1-x^2)^{-3/2}$

$\arccos(X)$	$\arccos(x)$	$-1/\sqrt{1-x^2}$	$-x\cdot(1-x^2)^{-3/2}$
$\mathrm{arctg}(X)$	$\mathrm{arctg}(x)$	$1/\,(1+x^2)$	$-2\cdot x/\,(1+x^2)^2$
$\mathrm{arcctg}(X)$	$\mathrm{arcctg}(x)$	$-1/\,(1+x^2)$	$2\cdot x/\,(1+x^2)^2$
$\mathrm{sh}(X)$	$\mathrm{sh}(x)$	$\mathrm{ch}(x)$	$\mathrm{sh}(x)$
$\mathrm{ch}(X)$	$\mathrm{ch}(x)$	$\mathrm{sh}\,(x)$	$\mathrm{ch}(x)$
$\mathrm{th}(X)$	$\mathrm{th}(x)$	$\mathrm{sech}^2(x)$	$-2\cdot\mathrm{sech}^2(x)\cdot\mathrm{th(x)}$
$\mathrm{cth}(X)$	$\mathrm{cth}(x)$	$-\mathrm{csch}^2(x)$	$2\cdot\mathrm{csch}^2(x)\cdot\mathrm{cth(x)}$
$\mathrm{sech}(X)$	$\mathrm{sech}(x)$	$-\mathrm{sech}(x)\cdot\mathrm{th(x)}$	$\mathrm{sh}^{-1}(x)\cdot(2\cdot\mathrm{ch}^2(x)\cdot\mathrm{sh}^{-2}(x)-1)$
$\mathrm{csch}(X)$	$\mathrm{csch}(x)$	$-\mathrm{csch}(x)\cdot\mathrm{cth(x)}$	$\mathrm{ch}^{-1}(x)\cdot(2\cdot\mathrm{sh}^2(x)\cdot\mathrm{ch}^{-2}(x)-1)$

Табл. 3. Компоненты основных элементарных функций усечённого гипер-дуального аргумента

Вычисление сложной дуальной функции (function composition) вида $F = f_1(f_2(\dots f_n(X)\dots),X),X)$ (где значение f_n используется в качестве аргумента для f_{n-1}) необходимо начать с вычисления $F_n = f_n(X)$, продолжить вычислениями $F_{n-1} = f_{n-1}(F_n,X)$, $F_{n-2} = f_{n-2}(F_{n-1},X)$, … , $F = f_1(F_2,X)$.

Для многих целей усечённое гипер-дуальное число $A = a + a_1\boldsymbol{\varepsilon} + a_2\boldsymbol{\omega}$ удобно представлять в виде матриц 3x3:

$$\boldsymbol{A} = \begin{vmatrix} a & a_1 & a_2 \\ 0 & a & a_1 \\ 0 & 0 & a \end{vmatrix}, \tag{3.5}$$

где элементы главной диагонали суть – главная часть $Re(A)$, а элементы (0, 1) и (1, 2) – $Im_1(A)$, (0, 2) – $Im_2(A)$. Все стандартные матричные операции применимы к матрице типа $\boldsymbol{A}$. Так, например, произведение матриц $\boldsymbol{A}$ и $\boldsymbol{B}$:

$$\boldsymbol{A}\cdot\boldsymbol{B} = \begin{vmatrix} a & a_1 & a_2 \\ 0 & a & a_1 \\ 0 & 0 & a \end{vmatrix}\cdot\begin{vmatrix} b & b_1 & b_2 \\ 0 & b & b_1 \\ 0 & 0 & b \end{vmatrix} = \begin{vmatrix} a\cdot b & a\cdot b_1 + b\cdot a_1 & a\cdot b_2 + a_1\cdot b_1 + a_2\cdot b \\ 0 & a\cdot b & a\cdot b_1 + b\cdot a_1 \\ 0 & 0 & a\cdot b \end{vmatrix}, \tag{3.6}$$

а из соотношения

$$\boldsymbol{A}\cdot\boldsymbol{A}^{-1} = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}, \tag{3.7}$$

найдем матрицу обратную $\boldsymbol{A}$**:**

$$\boldsymbol{A^{-1}} = \begin{vmatrix} a^{-1} & -a_1\cdot a^{-2} & {a_1}^2\cdot a^{-3} - a_2\cdot a^{-2} \\ 0 & a^{-1} & -a_1\cdot a^{-2} \\ 0 & 0 & a^{-1} \end{vmatrix} \tag{3.8}$$

4. Использование усечённых гипер-дуальных чисел в итерационных методах

Рассмотрим четыре итерационных метода решения нелинейных уравнений $f(x) = 0$: метод Ньютона [8], метод Чебышева [9], метод Рафсона [8] и непрерывный аналог метода Ньютона со специальной процедурой ускорения сходимости [10]. Итерационные формулы этих методов имеют следующий вид ($n = 0, 1, 2, ...$):

I. метод Ньютона:

$$x_{n+1} = x_n - a_1(x_n), \tag{4.1}$$

II. метод Чебышева третьего порядка с удержанием первых и вторых производных:

$$x_{n+1} = x_n - a_1(x_n)\cdot[1 + 0.5\cdot a_2(x_n)\cdot(1 + a_2(x_n))], \tag{4.2}$$

III. метод Рафсона:

$$x_{n+1} = x_n - a_1(x_n)\cdot[1 - 0.5\cdot a_2(x_n)]^{-1}, \tag{4.3}$$

IV. непрерывный аналог метода Ньютона:

$$x_{n+1} = s_n - a_1(s_n), \quad s_{n+1} = x_{n+1} - a_1(x_{n+1}), \tag{4.4}$$

где: $a_1(x) = f(x)/f'(x)$ и $a_2(x) = a_1(x)\cdot f''(x)/f'(x)$.

Из формул (4.1) ÷ (4.4) видно, что на каждом итерационном шаге нужно вычислять значения функции и ее производных: метод Ньютон – $f(x)$ и $f'(x)$; метод Чебышева и Рафсона – $f(x)$, $f'(x)$, $f''(x)$; непрерывный аналог метода Ньютона – $f(x)$, $f'(x)$, $f(s)$, $f'(s)$. Учет производных функции $f(x)$ приводит к увеличению скорости сходимости, но требует вычисления этих производных, что может оказаться неприемлемым или труднодостижимым. Однако, если использовать метод автоматического дифференцирования (AD), построенный на базе усечённых гипер-дуальных чисел, то за одно обращение к $F(X)$ получаем $f(x)$, $f'(x)$ и $f''(x)$. В этом случае на каждом итерационном шаге методы (I), (II) и (III) потребуют одно обращение к $F(X)$, а метод (IV) два обращения. При этом значения производных будут вычислены с точностью вычисления элементарных функций (табл. 3).

Применим теперь приведенные выше соотношения (4.1) ÷ (4.4) для итерационных процессов с использованием усечённых гипер-дуальных функций и переменных. На первом этапе надо заменить исследуемую функцию действительного аргумента $f(x)$ на соответствующую эквивалентную функции $F(X)$ усечённого гипер-дуального аргумента. Задать начальное приближение x_0. Тогда значение F_n, соответствующее $(n + 1)$-му шагу, определится по формуле

$$F_n = F(X_n) = F(x_n + 1\cdot\boldsymbol{\varepsilon} + 0\cdot\boldsymbol{\omega}) = f(x_n) + f'(x_n)\cdot\boldsymbol{\varepsilon} + f''(x_n)\cdot\boldsymbol{\omega} \tag{4.5}$$

Зная $f(x_n)$, $f'(x_n)$ и $f''(x_n)$ нетрудно определить $a_1(x_n)$ и $a_2(x_n)$ и по формулам (4.1) ÷ (4.4) найти X_{n+1} для следующей итерации. Для метода (IV) начальное значение $s_0 = x_0 - a_1(x_0)$.

Возможны разные критерии окончания итерационного процесса, например, $\|Re(F(X_n))\| \leq \delta$ – заданная малая величина.

5. Компьютерная реализация

Компьютерная реализация усечённых гипер-дуальных чисел была выполнена на языке SWIFT операционной системы macOS (см. Приложение 1). Был создан новый тип данных Thdn (truncated-dual number) и его расширение, которое включает перезагрузку элементарных функции (см. табл. 3) и операций.

Отдельно добавлены процедуры решения выше рассмотренных вычислительных задач: метод (I) реализует функция Newton(.); метод (II) – Chebyshev(.); метод (III) – Raphson(.); метод (IV) – NewtonEx(.). В Приложение 2 приводятся некоторые манипуляции с переменными и функциями Thdn типа. Программный код для исследования сходимости методов (I) ÷ (IV) представлен в Приложении 3.

6. Численные эксперименты

В работе [10] представлен непрерывный аналог метода Ньютона со специальной процедурой ускорения сходимости и проведено сравнение (по числу итераций) с известными итерационными процессами со сходимостью третьего и четвёртого порядка на базе восьми функций, приведенных в Приложении 4. Исследовалось восемь итерационных схем. Лучшей (по числу итераций) оказалась схема непрерывного аналога метода Ньютона со специальной процедурой ускорения сходимости (см. формулу (4.4)). Нами же был проведен численный анализ этой схемы и схем, определенных формулами (4.1) – (4.3), которые были реализованы на основе автоматического дифференцирования с использованием усечённых гипер-дуальных чисел. Результаты этого эксперимента (см. табл. 4) показали, что скорости сходимости (по числу итераций) методов II, III и IV достаточно близкие. Однако методы II и III требуют одно обращение к исследуемой гипер-дуальной функции, а метод IV – два, что приводит к увеличению вычислительного времени итерационного шага.

	f(x)	x_0	Корень	Количество итераций			
				I	II	III	IV
1.	$x^3 + 4x^2 - 10$	1.0	1.365230013414097	5	3	3	3
2.	$\sin^2(x) - x^2 + 1$	1.0	1.404491648215341	6	4	4	3
3.	$x^2 - e^x - 3x + 1$	3.0	0.257530285439861	6	4	4	3
4.	$\cos(x) - x$	1.0	0.739085133215161	4	3	3	2
5.	$(x-1)^3 - 1$	2.5	2.000000000000000	6	3	4	3
6.	$x^3 - 10$	1.5	2.154434690031884	6	4	4	3
7.	$xe^{x^2} - \sin^2(x) + 3\cos(x)$	-2.0	-1.207647827130919	8	5	4	4
8.	$e^{x^2+7x-30} - 1$	5.5	3.000000000000000	44	22	22	22

Табл. 4. Результаты тестирования исследуемых методов: I-Ньютон, II-Чебышев, III-Рафсон, IV- аналог Ньютона

Следует отметить, что метод III (Рафсона) менее чувствителен к начальному приближению чем все остальные. В табл. 5 приведены результаты тестирования с иными начальными приближениями чем в табл. 4 (прочерк означает, что процесс расходится).

	f(x)	x_0	Корень	Количество итераций			
				I	II	III	IV
1.	$x^3 + 4x^2 - 10$	-1.0	1.365230013414097	24	45	10	12
5.	$(x-1)^3 - 1$	0.0	2.000000000000000	9	5	1	5
7.	$xe^{x^2} - \sin^2(x) + 3\cos(x)$	1.0	-1.207647827130919	7	-	10	4
8.	$e^{x^2+7x-30} - 1$	2.5	3.000000000000000	-	-	6	-

Табл. 5. Влияние x_0 на сходимость методов: I-Ньютон, II-Чебышев, III-Рафсон, IV- аналог Ньютона

6. Заключение

Получен новый тип гипер-дуальных чисел – усечённые гипер-дуальные числа (Thdn). По сравнению с гипер-дуальными числами, компьютерная реализация этого типа данных (Thdn) позволяет сэкономить память. Результатом обращения к усечённой гипер-дуальной функции являются точные значения самой функции и ее производных (без дублирования значений). Компьютерная реализация этого типа данных оказалась достаточно компактной. Программный код для анализа скорости сходимости исследуемых итерационных методов решения нелинейных уравнений доступен в Приложении 1. Заметим, что основной целью работы не было фундаментальное исследование сходимости итерационных методов. Главная задача – продемонстрировать сравнительную простоту, компактность, точность, надежность и универсальность описанного подхода в решении итерационных задач с учетом производных первого и второго порядков.

Приложение 1

Описание типа данных Thdn (truncated-hyper-dual number) для Swift 5 (macOS 10.14)

```
import Foundation

precedencegroup SuperPowerPrecedence {
    associativity: left
    assignment: true
    higherThan: MultiplicationPrecedence
}
```

```
infix operator **:SuperPowerPrecedence

struct Thdn{
    var re,im1,Im2: Double;
    //--Initializers:
    init() {...}
    init(re:Double) {...}
    init(re:Double,im1:Double,im2:Double) {...}
    //--Methods:
    func norm() -> Double {...}
    func string() -> String {...}
}
extension Thdn {
    //--CONSTANTS:
    static let accuracy:Double = 1e-15, PI:Double = Double.pi;
    //--OPERATORS (overloading):
    static prefix func - ( A:Thdn) -> Thdn {...}
    static prefix func + ( A:Thdn) -> Thdn {...}
    static func +  (A:Thdn, B:Thdn) -> Thdn {...}
    static func += (lhs: inout Thdn, rhs: Thdn) {...}
    static func -  (A:Thdn, B:Thdn) -> Thdn {...}
    static func -= (lhs: inout Thdn, rhs: Thdn) {...}
    static func *  (A: Thdn, B: Thdn) -> Thdn {...}
    static func *  (A: Thdn, B: Double) -> Thdn {...}
    static func *  (A: Double, B: Thdn) -> Thdn {...}
    static func *= (lhs: inout Thdn, rhs: Thdn) {...}
    static func *= (lhs: inout Thdn, rhs: Double) {...}
    static func /  (A: Thdn, B: Thdn) -> Thdn {...}
    static func /  (A: Thdn, B: Double) -> Thdn {...}
    static func ** (left: Thdn, right: Double) -> Thdn {...}
    static func /= (lhs: inout Thdn, rhs: Thdn) {...}
    static func /= (lhs: inout Thdn, rhs: Double) {...}
    static func == (left: Thdn,  right: Thdn) -> Bool {...}
    static func != (left: Thdn,  right: Thdn) -> Bool {...}
    static func <  (left: Thdn,  right: Thdn) -> Bool {...}
    static func >  (left: Thdn,  right: Thdn) -> Bool {...}
    static func <= (left: Thdn, right: Thdn) -> Bool {...}
    static func >= (left: Thdn, right: Thdn) -> Bool {...}
    //--FUNCTIONS:
    static func inverse(A:Thdn) -> Thdn {...}
    static func pow(x:Double, n:Double) -> Thdn {...}
    static func pow(X:Thdn, n:Double) -> Thdn {...}
    static func powX(a:Double, x:Double) -> Thdn {...}
```

```
static func log(x:Double) -> Thdn {...}
static func log(X:Thdn) -> Thdn {...}
static func sqrt(x:Double) -> Thdn {...}
static func sqrt(X:Thdn) -> Thdn {...}
static func exp(x:Double) -> Thdn {...}
static func exp(X:Thdn) -> Thdn {...}
static func sin(x:Double) -> Thdn {...}
static func sin(X:Thdn) -> Thdn {...}
static func cos(x:Double) -> Thdn {...}
static func cos(X:Thdn) ->Thdn {...}
static func tan(x:Double) -> Thdn {...}
static func tan(X:Thdn) -> Thdn {...}
static func asin(x:Double) -> Thdn {...}
static func asin(X:Thdn) -> Thdn {...}
static func acos(x:Double) -> Thdn {...}
static func acos(X:Thdn) -> Thdn {...}
static func atan(x:Double) -> {...}
static func atan(X:Thdn) -> Thdn {...}
static func max(X1:Thdn, X2:Thdn) -> Thdn {...}
static func min(X1:Thdn, X2:Thdn) -> Thdn {...}
//--SOLUTIONS:
static func Newton (x0:Double, F:(Thdn)->Thdn, E:Double, maxIteraction:Int=100)->(n:Int, x:Double){
    var Xn = x0, Fn: Thdn, i:Int = 0;
    while true {
        Fn =  F(Thdn (re: Xn, im1: 1, im2: 0));
        if abs(Fn.re) <= E || i > maxIteraction || Xn.isNaN {return (n:i, x:Xn)}
        Xn = Xn - Fn.re/Fn.im1;   i += 1;
    }
    return (n:i, x:Double.infinity);
}

static func Chebyshev (x0:Double, F:( Thdn)-> Thdn, E:Double, maxIteraction:Int=100)->(n:Int, x:Double){
    var Xn = x0,  Fn: Thdn,  i:Int = 0, a1:Double=0, a2:Double;
    while true {
        Fn =  F(Thdn (re: Xn, im1: 1, im2: 0));
        if abs(Fn.re) <= E || i > maxIteraction || Xn.isNaN {return (n:i, x:Xn)}
        a1 = Fn.re/Fn.im1;  a2 = a1*Fn.im2/Fn.im1
        Xn = Xn - a1*(1.0 + 0.5*a2*(1.0 + a2);   i += 1;
    }
    return (n:i, x:Double.infinity);
}

static func Raphson (x0:Double, F:(Thdn)-> Thdn, E:Double, maxIteraction:Int=100)->(n:Int, x:Double){
```

```
    var Xn = x0,  Fn:Thdn, i:Int = 0, a1:Double=0, a2:Double;
      while true {
         Fn =  F(Thdn(re: Xn, im1: 1, im2: 0));
         if abs(Fn.re) <= E || i > maxIteraction || Xn.isNaN {return (n:i, x:Xn)}
         a1 = Fn.re/Fn.im1;  a2 = a1*Fn.im2/Fn.im1;
         Xn = Xn - a1/(1 - 0.5*a2);   i += 1;
      }
    return (n:i, x:Double.infinity);
  }

  static func NewtonEx (x0:Double, F:(Thdn)-> Thdn, E:Double, maxIteraction:Int=100)->(n:Int, x:Double){
      var Xn = x0,  Fn:Thdn, i:Int = 1, Sn:Double = 0;
      Fn =  F(Thdn(re: Xn, im1: 1, im2: 0));    Sn = Xn - Fn.re/Fn.im1;
      while true {
         Fn =  F(Thdn(re: Sn, im1: 1, im2: 0));  Xn = Sn - Fn.re/Fn.im1;
         Fn =  F(Thdn(re: Xn, im1: 1, im2: 0));
         if abs(Fn.re) <= E || i > maxIteraction || Xn.isNaN {return (n:i, x:Xn)}
         Sn = Xn - Fn.re/Fn.im1;   i += 1;
      }
      return (n:i, x:Double.infinity);
  }
```

Приложение 2

Результаты манипуляций (тест-драйв) с усечёнными гипер-дуальными числами типа Thdn (см. *Приложение 1*)

```
//--Operators:
      var A,B,C: Thdn
      A = Thdn (re: 1, im1: 2, im2: 3)       // A = 1.0  + 2.0•ε  + 3.0•ω
      B = Thdn (re: 4, im1: 5, im2: 6)       // B = 4.0  + 5.0•ε  + 6.0•ω
      C = A + B                    // C = 5.0  + 7.0•ε  + 9.0•ω
      C = A*B                      // C = 4.0  + 13.0•ε + 38.0•ω
      C = Thdn.inverse(A: A)                 // C = 1.0  - 2.0•ε  + 5.0•ω
      C = A/B                      // C = 0.25 + 0.1875•ε - 0.09375•ω

//--Basic functions:
      var Y: Thdn = Thdn.exp(x: 0)           // Y = 1 + 1•ε + 1•ω
```

```
Y = Thdn.cos(x: 0)              // Y = 1 + 0•ε - 1•ω
Y = Thdn.pow(x: 2, n: 3)        // Y = 8 + 12•ε + 12•ω
Y = Thdn.sqrt(x: 4)             // Y = 2 + 0.25•ε - 0.03125•ω
Y = Thdn.log(x: 1)              // Y = 0 + 1•ε - 1•ω
```

Приложение 3

Исследование сходимости методов: (I) – Newton(.); (II) – Chebyshev(.); (III) – Raphson(.); (IV) – NewtonEx(.) на восьми функциях.

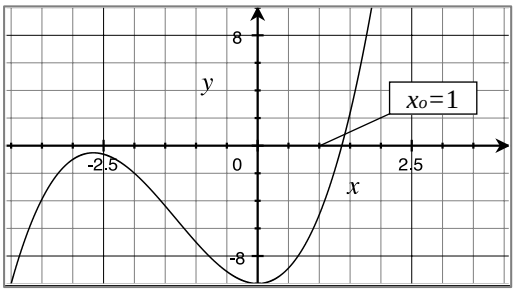
```
func f1(X: Thdn)-> Thdn {return Thdn.pow(X: X, n: 3) + 4* Thdn.pow(X: X, n: 2 ) - Thdn(re:10)};
func f2(X: Thdn)-> Thdn {return Thdn.sin(X: X)* Thdn.sin(X: X) - X*X + Thdn(re:1)};
func f3(X: Thdn)-> Thdn {return  X*X - Thdn.exp(X: X)  - 3*X + Thdn(re:2)};
func f4(X: Thdn)-> Thdn {return  Thdn.cos(X: X) - X};
func f5(X: Thdn)-> Thdn {return  Thdn.pow(X: X - Thdn(re:1), n: 3) - Thdn (re:1)};
func f6(X: Thdn)-> Thdn {return  Thdn.pow(X: X, n: 3) - Thdn(re:10)}
func f7(X: Thdn)-> Thdn {return  X*Thdn.exp(X: X*X) - Thdn.sin(X: X)* Thdn.sin(X: X)
                    + 3*Thdn.cos(X: X) +  Thdn(re:5)};
func f8(X: Thdn)-> Thdn {return  Thdn.exp(X: X*X + 7*X - Thdn (re:30)) - Thdn (re:1)};

let funcs = [f1, f2, f3, f4, f5, f6, f7, f8];
let x0:[Double] = [1, 1, 3, 1, 2.5, 1.5, -2.0, 5.5];
var x:(n: Int, x: Double);
for i in 0...7 {
      print ("Function f\(i+1)");
   x = Thdn.Newton(x0: x0[i], F: funcs[i], E: 1e-14);
      print ("    Newton:     n = \(x.n); x = \(x.x)");
   x = Thdn.Chebyshev(x0: x0[i], F: funcs[i], E: 1e-14);
      print ("    Chebyshev:  n = \(x.n); x = \(x.x)");
   x = Thdn.Raphson(x0: x0[i], F: funcs[i], E: 1e-14);
      print ("    Raphson:    n = \(x.n); x = \(x.x)");
   x = Thdn.NewtonEx(x0: x0[i], F: funcs[i], E: 1e-14);
      print ("    NewtonEx:   n = \(x.n); x = \(x.x)");
}

//-- OUTPUT WITH FRACTIONAL PART OF 15 DIGITS

Function f1
            Newton:     n = 5; x = 1.365230013414097
            Chebyshev:  n = 3; x = 1.365230013414097
```

```
        Raphson:    n = 3; x = 1.365230013414097
        NewtonEx:   n = 3; x = 1.365230013414097
Function f2
        Newton:     n = 6; x = 1.404491648215341
        Chebyshev:  n = 4; x = 1.404491648215341
        Raphson:    n = 4; x = 1.404491648215341
        NewtonEx:   n = 3; x = 1.404491648215341
Function f3
        Newton:     n = 6; x = 0.257530285439861
        Chebyshev:  n = 4; x = 0.257530285439861
        Raphson:    n = 4; x = 0.257530285439861
        NewtonEx:   n = 3; x = 0.257530285439861
Function f4
        Newton:     n = 4; x = 0.739085133215161
        Chebyshev:  n = 3; x = 0.739085133215161
        Raphson:    n = 3; x = 0.739085133215161
        NewtonEx:   n = 2; x = 0.739085133215161
Function f5
        Newton:     n = 6; x = 2.000000000000000
        Chebyshev:  n = 3; x = 2.000000000000000
        Raphson:    n = 4; x = 2.000000000000000
        NewtonEx:   n = 3; x = 2.000000000000000
Function f6
        Newton:     n = 6; x = 2.154434690031884
        Chebyshev:  n = 4; x = 2.154434690031884
        Raphson:    n = 4; x = 2.154434690031884
        NewtonEx:   n = 3; x = 2.154434690031884
Function f7
        Newton:     n = 8; x = -1.207647827130919
        Chebyshev:  n = 5; x = -1.207647827130919
        Raphson:    n = 4; x = -1.207647827130919
        NewtonEx:   n = 4; x = -1.207647827130919
Function f8
        Newton:     n = 44; x = 3.000000000000000
        Chebyshev:  n = 22; x = 3.000000000000000
        Raphson:    n = 22; x = 3.000000000000000
      NewtonEx:   n = 22; x = 3.000000000000000
```

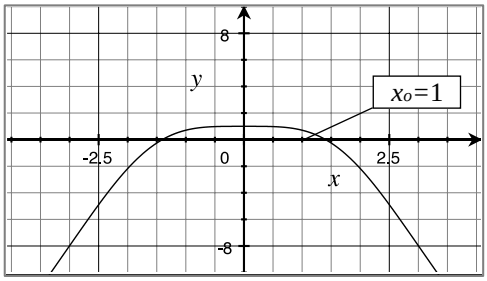
Приложение 4

Графики исследуемых функций

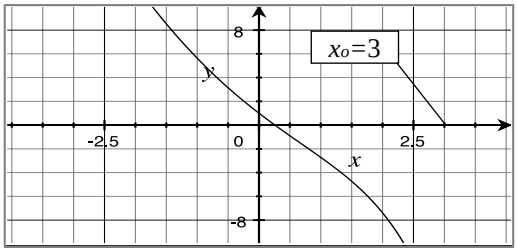
1) $f(x) = x^3 + 4x^2 - 10$



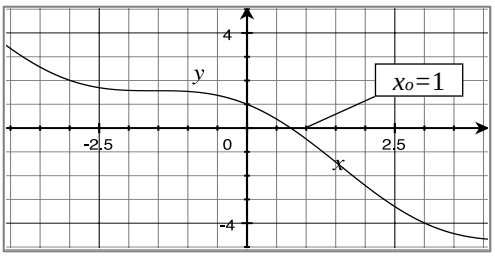
2) $f(x) = \sin^2(x) - x^2 + 1$



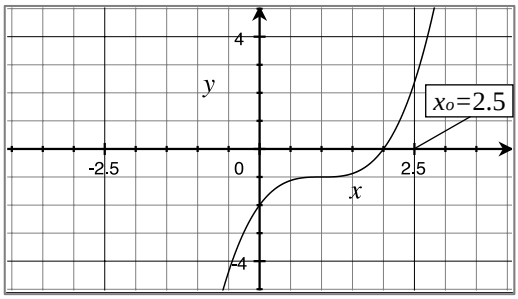
3) $f(x) = x^2 - e^x - 3x + 1$



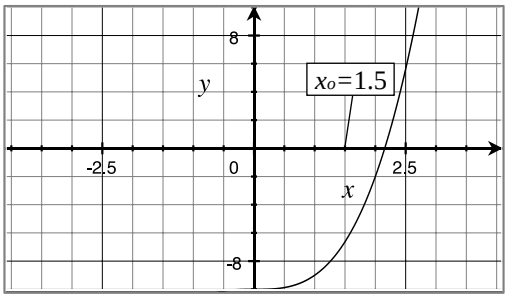
4) $f(x) = \cos(x) - x$



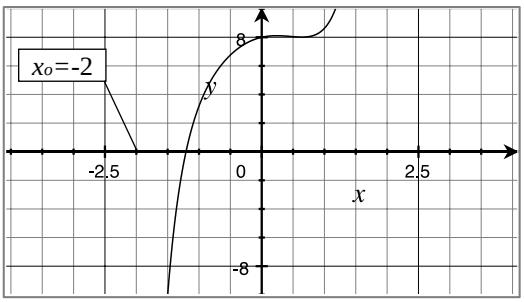
5) $f(x) = (x-1)^3 - 1$



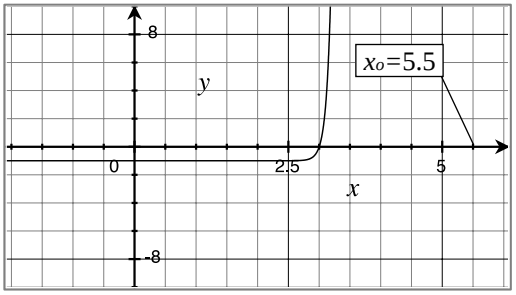
6) $f(x) = x^3 - 10$



7) $f(x) = xe^{x^2} - \sin^2(x) + 3\cos(x)$



8) $f(x) = e^{x^2+7x-30} - 1$



Литература

1. *А. Н. Тихонов, В. Я. Арсенин.* Методы решения некорректных задач. – М.: Наука,1979. – 285с.
2. *U. Naumann*. The Art of Differentiating Computer Programs. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2012.
3. *Яглом И. М.* Комплексные числа и их применение в геометрии. М.: Государственное изд-во физико-математической литературы, 1963. – 192 с.
4. *С. П. Шарый*. Курс вычислительных методов. – Новосибирск: Новосиб. гос. ун-т., 2014. - 507 с.
5. *J. A. Fike and J. J. Alonso.* The Development of Hyper-Dual Numbers for Exact Second Derivative Calculations. AIAA paper 2011-886, 49th AIAA Aerospace Sciences Meeting, January 4-7, 2011
6. *J. A. Fike.* Multi-Objective Optimization Using Hyper-Dual Numbers. Ph.D. Dissertation, Stanford University, 2013.
7. *Двайт Г. Б.* Таблицы интегралов и другие математические формулы. - М.: Наука, 1973. - 228 с.
8. *М. А. Тынкевич, А. Г. Пимонов*. Введение в численный анализ. - КузГТУ. – Кемерово, 2017. - 176 с.
9. *G. Antoni.* An Efficient and Straightforward Numerical Technique Coupled to Classical Newton's Method for Enhancing the Accuracy of Approximate Solutions Associated with Scalar Nonlinear Equations. International Journal of Engineering Mathematics, vol. 2016, Article ID 8565821, 12 p.
10. *Т. Жанлав, О. Чулуунбаатар.* О некоторых итерационных методах высокого порядка сходимости для решения нелинейных уравнений. - Вестник РУДН. Серия Математика. Информатика. Физика. No 4. 2009. С. 47–55

Абстракт

В данной статье рассматриваются вопросы, связанные с применением специальных чисел (гипер-дуальных чисел) в методе компьютерного автоматического дифференцирования. Вводится новый тип чисел (усечённые гипер-дуальные числа), которые свободны от избыточности, присущей гипер-дуальным числам. Приводятся основные операции и базовые функции пространства усечённых гипер-дуальных чисел. Дана их матричная форма представления, примеры реализации итерационных методов Ньютона, Чебышева, Рафсона и непрерывного аналога метода Ньютона со специальной процедурой ускорения сходимости на основе усечённых гипер-дуальных функций. В приложениях 1, 2 и 3 приведена компьютерная реализация усечённых гипер-дуальных чисел для языка SWIFT операционной системы macOS.

Ключевые слова*: гипер-дуальные числа, усеченные гипер-дуальные числа, автоматическое дифференцирование, нелинейные уравнения, итерационные методы.*

23 мая 2019 г.