УСЕЧЕННЫЕ ГИПЕР-ДУАЛЬНЫЕ ЧИСЛА В АВТОМАТИЧЕСКОМ ДИФФЕРЕНЦИРОВАНИИ

Олифер В.И.

Точность вычислений является основным критерием качества результатов вычислений. Любой вычислительный процесс содержит погрешность метода или усечения (truncation error) и погрешность конечного округления (finite precision). 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: $\mathcal{O}(k)$ 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 автоматическое дифференцирование (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].

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

×	**1**	$\boldsymbol{\varepsilon}_1$	$\boldsymbol{\varepsilon}_2$	$\boldsymbol{\varepsilon}_{12}$
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

(1)

Таблица 1. Правила умножения элементов базиса гипер-дуальных чисел

При этом гипер-дуальное число представимо в виде:

$$\mathcal{X} = x + x_1\boldsymbol{\varepsilon}_1 + x_2\boldsymbol{\varepsilon}_2 + x_3\boldsymbol{\varepsilon}_{12}\,, \tag{2}$$

а гипер-дуальная функция определяется выражением:

$$F(\mathcal{X}) = f(x) + x_1{\cdot}f'(x)\boldsymbol{\varepsilon}_1 + x_2{\cdot}f'(x)\boldsymbol{\varepsilon}_2 + (x_3{\cdot}f'(x) + x_1{\cdot}x_2{\cdot}f''(x))\boldsymbol{\varepsilon}_{12} \tag{3}$$

В соотношениях (1) ÷ (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:

$$F(\mathcal{X}) = f(x) + f'(x)\boldsymbol{\varepsilon}_1 + f'(x)\boldsymbol{\varepsilon}_2 + f''(x)\boldsymbol{\varepsilon}_{12} \tag{4}$$

Для итерационных процессов, использующих значения функции, ее первой и второй производных, выражение (4) является избыточным т. к. первая и вторая мнимые части содержат одну и туже величину – значение первой производной. Чтобы избавиться от указанной избыточности введем базис $\{\mathbf{1},\ \boldsymbol{\varepsilon},\ \boldsymbol{\omega}\}$ с правилом умножения:

×	***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*

(5)

Таблица 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

$$F(X) = f(x) + x_1{\cdot}f'(x)\boldsymbol{\varepsilon} + (x_2{\cdot}f'(x) + x_1^2{\cdot}f''(x))\boldsymbol{\omega} \tag{6}$$

При $x_1 = 1$ и $x_2 = 0$ выражение (6) принимает вид:

$$F(X) = f(x) + f'(x)\boldsymbol{\varepsilon} + f''(x)\boldsymbol{\omega} \tag{7}$$

Полученное соотношение (7) свободно от избыточности, присутствующей в (4).
Алгебраические операции сложения, умножения, обращения и деления (с учетом Таблицы 2) определены по правилам:

$$
\begin{aligned}
&A = a + a_1\boldsymbol{\varepsilon} + a_2\boldsymbol{\omega}, \qquad B = b + b_1\boldsymbol{\varepsilon} + b_2\boldsymbol{\omega}\\
&A + B = a + b + (a_1 + b_1)\boldsymbol{\varepsilon} + (a_2 + b_2)\boldsymbol{\omega},\\
&A{\cdot}B = a{\cdot}b + (a{\cdot}b_1 + b{\cdot}a_1)\boldsymbol{\varepsilon} + (a{\cdot}b_2 + 2{\cdot}a_1{\cdot}b_1 + b{\cdot}a_2)\boldsymbol{\omega},\\
&A^{-1} = a^{-1} - a_1{\cdot}a^{-2}\boldsymbol{\varepsilon} + (2{\cdot}a_1^2{\cdot}a^{-3} - a_2{\cdot}a^{-2})\boldsymbol{\omega},\\
&{}^{\mathrm{A}}/_{B} = \mathrm{A}{\cdot}\mathrm{B}^{-1} = a{\cdot}b^{-1} + (a_1{\cdot}b^{-1} - a{\cdot}b_1{\cdot}b^{-2})\boldsymbol{\varepsilon} + [2{\cdot}(a{\cdot}b_1^2{\cdot}b^{-3} - a_1{\cdot}b_1{\cdot}b^{-2}) -\\
&a{\cdot}b_2{\cdot}b^{-2} + a_2{\cdot}b^{-1}]\boldsymbol{\omega}
\end{aligned}
\qquad (8)
$$

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

$F(\boldsymbol{X})$	$\boldsymbol{f(x)}$	$\boldsymbol{f'(x)}$	$\boldsymbol{f''(x)}$
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}	$-x^{-2}$
$\log_a(X)$	$\log_a(x)$	$(x\ln(a))^{-1}$	$-x^{-2}{\cdot}\ln(a)^{-1}$
$\sin(X)$	$\sin(x)$	$\cos(x)$	$-\sin(x)$
$\cos(X)$	$\cos(x)$	$-\sin(x)$	$-\cos(x)$
$\mathrm{tg}(X)$	$\mathrm{tg}(x)$	$\sec^2(x) = \cos^{-2}(x)$	$2{\cdot}\mathrm{tg}(x){\cdot}\cos^{-2}(x)$
$\mathrm{ctg}(X)$	$\mathrm{ctg}(x)$	$-\csc^2(x) = -\sin^{-2}(x)$	$-2{\cdot}\mathrm{ctg}(x){\cdot}\sin^{-2}(x)$
$\sec(X)$	$\sec(x)$	$\sec(x){\cdot}\,\mathrm{tg}(x)$	$\sec(x){\cdot}(2{\cdot}\mathrm{tg}^2(x)+1)$
$\csc(X)$	$\csc(x)$	$-\csc(x){\cdot}\,\mathrm{ctg}(x)$	$\csc(x){\cdot}(2{\cdot}\mathrm{ctg}^2(x)+1)$
$\arcsin(X)$	$\arcsin(x)$	$1/\sqrt{1-x^2}$	$x{\cdot}(1-x^2)^{-3/2}$
$\arccos(X)$	$\arccos(x)$	$-1/\sqrt{1-x^2}$	$-x{\cdot}(1-x^2)^{-3/2}$
$\mathrm{arctg}(X)$	$\mathrm{arctg}(x)$	$1/(1+x^2)$	$-2{\cdot}x/(1+x^2)^2$
$\mathrm{arcctg}(X)$	$\mathrm{arcctg}(x)$	$-1/(1+x^2)$	$2{\cdot}x/(1+x^2)^2$
$\mathrm{sh}(X)$	$\mathrm{sh}(x)$	$\mathrm{ch}(x)$	$\mathrm{sh}(x)$
$\mathrm{ch}(X)$	$\mathrm{ch}(x)$	$\mathrm{sh}(x)$	$\mathrm{ch}(x)$
$\mathrm{th}(X)$	$\mathrm{th}(x)$	$\mathrm{sech}^2(x)$	$-2{\cdot}\mathrm{sech}^2(x){\cdot}\mathrm{th}(x)$
$\mathrm{cth}(X)$	$\mathrm{cth}(x)$	$-\mathrm{csch}^2(x)$	$2{\cdot}\mathrm{csch}^2(x){\cdot}\mathrm{cth}(x)$
$\mathrm{sech}(X)$	$\mathrm{sech}(x)$	$-\mathrm{sech}(x){\cdot}\mathrm{th}(x)$	$\mathrm{sh}^{-1}(x){\cdot}(2{\cdot}\,\mathrm{ch}^2(x){\cdot}\,\mathrm{sh}^{-2}(x)-1)$
$\mathrm{csch}(X)$	$\mathrm{csch}(x)$	$-\mathrm{csch}(x){\cdot}\mathrm{cth}(x)$	$\mathrm{ch}^{-1}(x){\cdot}(2{\cdot}\,\mathrm{sh}^2(x){\cdot}\,\mathrm{ch}^{-2}(x)-1)$

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

Вычисление сложной дуальной функции (function composition) вида $F = f_1(f_2(\ldots f_n(X)\ldots), X)$ (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:

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

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 $\boldsymbol{B}$:

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

а из соотношения

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

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

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

Рассмотрим некоторые практические задачи. Численное решение нелинейных (алгебраических или трансцендентных) уравнений вида $f(x) = 0$ осуществляется с помощью итерационных алгоритмов, уточняющих решение на каждом шаге итерации по формуле $x_{n+1} = x_n - g(x,\ f(x),\ f'(x),\ f''(x),\ \ldots)|_{x=x_n}$, где x_n решение полученное на n-ом шаге итерационного процесса. От вида функции $g(\ldots)$ зависит скорость сходимости и сложность реализации итерационного алгоритма. По скорости сходимости различают алгоритмы с линейной, квадратичной, кубической и т. д. скоростями сходимости. Наличие производных функции $f(x)$ различных порядков в $g(\ldots)$ определяют скорость сходимости. Таким образом, учет производных функции $f(x)$ приводит к увеличению скорости сходимости, но требует вычисления этих производных, что может оказаться неприемлемым или труднодостижимым. Однако в этом случае неоспоримую услугу может оказать метод автоматического дифференцирования (AD), построенный на базе усечённых гипер-дуальных чисел.
Как правило, функция $g(\ldots)$ соответствует формуле Чебышева [4]:

$$g(\ldots) = \frac{f(x)}{f'(x)}\left[1 + \frac{1}{2}\cdot\frac{f(x)\cdot f''(x)}{f'(x)} + \frac{f(x)^2}{f'(x)^2}\left[\frac{1}{2}\cdot\left(\frac{f''(x)}{f'(x)}\right)^2 - \frac{1}{6}\cdot\frac{f'''(x)}{f'(x)}\right] + \cdots\right] \qquad (9)$$

На практике используются только первые два члена выше приведенного выражения и в этом случае новое приближение вычисляется по формуле:

$$x_{n+1} = x_n - \frac{f(x)}{f'(x)}\left[1 + \frac{1}{2}\cdot\frac{f(x)\cdot f''(x)}{f'(x)^2}\right] \qquad (10)$$

Полученное соотношение (10) описывает итерационный метод Ньютона-Чебышева (Newton-Chebyshev), который обладает кубической сходимостью [8]. Если ограничится только первым членом в квадратных скобках, то получим итерационную формулу метода Ньютона: $x_{n+1} = x_n - f(x)/f'(x)$, имеющего квадратичную сходимость.

Применим теперь полученные выше соотношения (методы Ньютона и Ньютона-Чебышева) для итерационных процессов с использованием усечённых гипер-дуальных переменных. Как в первом, так и во втором случаях надо заменить функцию действительного аргумента $f(x)$ на соответствующую эквивалентную функции $F(X)$ усечённого гипер-дуального аргумента. Задать начальное приближение $X_0 = x_0 + 0\cdot\boldsymbol{\varepsilon} + 0\cdot\boldsymbol{\omega}$. Тогда приближение $n+1$ будет равно: для метода Ньютона $X_{n+1} = X_n - \tau(X_n)$ и для метода Ньютона-Чебышева $X_{n+1} = X_n - \tau(X_n) - \mu(X_n)$, где $\tau(X_n) = Re\big(F(X_n)\big)/Im_1\big(F(X_n)\big)$ и $\mu(X_n) = 0.5\cdot Re\big(F(X_n)\big)^2\cdot Im_2\big(F(X_n)\big)/Im_1\big(F(X_n)\big)^3$. Возможны разные критерии окончания итерационного процесса, например, $\|Re(X_{n+1}) - Re(X_n)\| \le \delta$ – заданная малая величина. Заметим, что для поиска экстремума функции $f(x)$, итерационная формула метода Ньютона принимает вид: $x_{n+1} = x_n - f'(x)/f''(x)$ и $\tau(X_n) = Im_1\big(F(X_n)\big)/Im_2\big(F(X_n)\big)$.

Выше рассмотренные два метода реализованы в Приложении 1 (`newton(.), newtonEx(.)`), а результаты вычислений для функции $f(x) = \cos(x) - x^3$ можно найти в Приложении 2.

Вопрос представления функции многих усечённых гипер-дуальных переменных заслуживает отдельного внимания. Одним из перспективных подходов может быть основан на суперпозиции функций одной усечённой гипер-дуальной переменной. Суперпозиционная теорема Колмогорова [9] (это решение 13-й проблемы Д. Гильберта) утверждает, что любую непрерывную функцию n переменных можно представить с помощью операций сложения, умножения и суперпозиции из непрерывных функций одной переменной. Однако, эта теорема не дает ответа на вопрос как это сделать. Даже частные случаи порождают системы "функций-монстров". Рассмотрим случай функции двух переменных. Если возможно разделить переменные функции $F(X,Y)$, т. е. представить $F(X,Y)$ в виде:

$$F(X,Y) = \sum_i \varphi_i(X)\cdot\psi_i(Y), \tag{11}$$

то задача существенно упрощается. Функции $\varphi_i(X)$ и $\psi_i(Y)$ определяются соотношением (6), а их произведение $\varphi_i(X)\cdot\psi_i(Y)$ с учетом (8) имеет вид (индекс i опущен):

$$\begin{aligned}\varphi(X)\cdot\psi(Y) = {} & \varphi(x)\cdot\psi(y) + [\,y_1\cdot\varphi(x)\cdot\psi'(y)(y) + x_1\cdot\psi(y)\cdot\varphi'(x)]\boldsymbol{\varepsilon} + \\ & +[\,\varphi(x)\cdot\big(y_2\cdot\psi'(y) + y_1^2\cdot\psi''(y)\big) + \psi(y)\big(x_2\cdot\varphi'(x) + x_1^2\cdot\varphi''(x)\big) + \\ & +2\,x_1\cdot y_1\cdot\varphi'(x)\cdot\psi'(y)]\boldsymbol{\omega}\end{aligned} \tag{12}$$

где $X = x + x_1\cdot\boldsymbol{\varepsilon} + x_2\cdot\boldsymbol{\omega}$, $Y = y + y_1\cdot\boldsymbol{\varepsilon} + y_2\cdot\boldsymbol{\omega}$.

Разные комбинации x_1, x_2, y_1 и y_2 позволяют сохранить в (6) и (10) те либо иные члены. Наиболее актуальные комбинации (фильтры) x_k и y_k (dX, dY и dXY) приведены в следующей таблице.

	$\boldsymbol{X, Y}$	$\boldsymbol{F(X,Y) = f(X)\cdot\varphi(Y)}$
dX	$X = (x,1,0),\quad Y = (y,0,0)$	$F(x,y) + F'_x(x,y)\boldsymbol{\varepsilon} + F''_x(x,y)\boldsymbol{\omega}$
dY	$X = (x,0,0),\quad Y = (y,1,0)$	$F(x,y) + F'_y(x,y)\boldsymbol{\varepsilon} + F''_y(x,y)\boldsymbol{\omega}$
dXY	$X = (x,1,0),\quad Y = (y,1,0)$	$F(x,y) + \big[F'_x(x,y) + F'_y(x,y)\big]\boldsymbol{\varepsilon} + \big[F''_x(x,y) + 2F''_{xy}(x,y) + F''_y(x,y)\big]\boldsymbol{\omega}$

Таблица 4. Актуальные фильтры dX, dY и dXY

Из Таблицы 4 видно, что главные части dX, dY и dXY содержать значение функции в точке (x, y), а мнимые части – частные производные:

- $Im1(dX) = F'_x(x,y)$, $Im2(dX) = F''_x(x,y)$
- $Im1(dY) = F'_y(x,y)$, $Im2(dY) = F''_y(x,y)$
- $Im1(dXY) = F'_x(x,y) + F'_y(x,y)$, $Im2(dXY) = F''_x(x,y) + 2F''_{xy}(x,y) + F''_y(x,y)$

Для определения $F''_{xy}(x,y)$ можно использовать формулу $F''_{xy}(x,y) = (Im2(dXY) - Im2(dX) - Im2(dY))/2$.

И, наконец, приведем выражения для вычисления элементов градиента *grad*(F) = {$Im1(dX)$, $Im1(dY)$} и гессиана *H*(F) = {$Im2(dX)$, $(Im2(dXY) - Im2(dX) - Im2(dY))/2$, $Im2(dY)$}.

Для функции двух переменных $f(x,y)$ метод Ньютона определяется формулой:

$$\begin{bmatrix} x \\ y \end{bmatrix}_{n+1} = \begin{bmatrix} x \\ y \end{bmatrix}_n - G^{-1}(x_n, y_n) \cdot \nabla F(x_n, y_n), \tag{13}$$

где: $G(x,y) = \begin{bmatrix} F''_x(x,y) & F''_{xy}(x,y) \\ F''_{yx}(x,y) & F''_y(x,y) \end{bmatrix}$ и $\nabla F(x,y) = \begin{bmatrix} F'_x(x,y) \\ F'_y(x,y) \end{bmatrix}$ – матрица Гессе и градиент функции $F(x,y)$ соответственно. Матрица обратная $G(x,y)$ имеет вид:

$$G^{-1}(x,y) = \det\big(G(x,y)\big) \cdot \begin{bmatrix} F''_y(x,y) & -F''_{xy}(x,y) \\ -F''_{yx}(x,y) & F''_x(x,y) \end{bmatrix}, \tag{14}$$

где $\det\big(G(x,y)\big) = 1/(F''_x(x,y) \cdot F''_y(x,y) - F''_{xy}(x,y) \cdot F''_{yx}(x,y))$.

Тогда окончательно имеем:

$$\begin{bmatrix} x \\ y \end{bmatrix}_{n+1} = \begin{bmatrix} x \\ y \end{bmatrix}_n - \det(x_n, y_n) \begin{bmatrix} F''_y(x,y) \cdot F'_x(x,y) - F''_{xy}(x,y) \cdot F'_y(x,y) \\ F''_x(x,y) \cdot F'_y(x,y) - F''_{yx}(x,y) \cdot F'_x(x,y) \end{bmatrix}_n. \tag{15}$$

Как видно, для вычисления следующего приближения $n+1$ необходимо знать: $F'_x(x_n, y_n)$, $F'_y(x_n, y_n)$, $F''_x(x_n, y_n)$, $F''_y(x_n, y_n)$, $F''_{xy}(x_n, y_n)$. Если $F(x,y)$ представима в виде $F(x,y) = \varphi(x) \cdot \psi(y)$, то как было показано выше $F'_x(x,y) = \varphi'(x) \cdot \psi(y)$, $F''_x(x,y) = \varphi''(x) \cdot \psi(y)$, $F'_y(x,y) = \varphi(x) \cdot \psi'(y)$, $F''_y(x,y) = \varphi(x) \cdot \psi''(y)$, $F''_{xy}(x,y) = \varphi'(x) \cdot \psi'(y)$ и можно воспользоваться таблицей элементарных (базовых) функций одного усечённого гипер-дуального аргумента (см. Таблицу 3).

Описанный метод реализован в Приложении 1 (`newtonXY(.)`), а результаты вычислений для функции $f(x) = 4x^2 + 3y^2 - 4xy + x$ можно найти в Приложении 2.

Заметим, что компьютерный процесс вычисления следующего приближения может быть ускорен, если для матричных операций использовать библиотеку SIMD (single instruction, multiple data), которая присутствует во многих языках программирования. При этом усечённые гипер-дуальные числа должны быть представлены в матричном виде.

Если аналитическое разделение переменных затруднительно, то можно воспользоваться, например, $\sum\Pi$ – аппроксимацией [10].

И, наконец, рассмотрим применение усечённых гипер-дуальных чисел в градиентной процедуре Парето оптимизации [6], которая относится к многоцелевой оптимизации некоторой векторной функции от векторного аргумента $\vec{f}(\vec{x})$. Каждая компонента $f_k(\vec{x})$

векторной функции $\vec{f}(\vec{x})$ является отдельной функцией цели, которые конкурируют между собой. Для простаты и наглядности дальнейшего изложения положим, что $\vec{f}$ состоит из двух компонент $f_1(x)$ и $f_2(x)$. Обычно несколько целей объединяются в одну цель при помощи весовых параметров (такой подход часто называется стратегией взвешенных сумм). В нашем случае новая единая целевая функция определяется как совокупная целевая функция $J(x) = (1 - \lambda)\cdot f_1(x) + \lambda \cdot f_2(x)$, где λ – весовой параметр $0 \le \lambda \le 1$. Условие оптимальности для $J(x)$ имеет вид:

$$J'(x) = (1 - \lambda)\cdot f_1'(x) + \lambda \cdot f_2'(x) = 0 \qquad (16)$$

Когда это условие выполняется, направление улучшения для одной цели $f_1(x)$ отрицательно влияет на другую цель $f_2(x)$ и наоборот. Следовательно, значения x, удовлетворяющие (16), являются оптимальными по Парето и называется многообразием Парето, которое может состоять из бесконечного числа оптимальных по Парето точек. На рисунке 1 отрезок оси *x* [*а*, *в*] является многообразием Парето. При этом точке *а* соответствует $\lambda = 0$, а точке *в* – $\lambda = 1$. Эти точки называются *якорными*. Большинство многоцелевых методов оптимизации пытаются найти только конечный набор из *N* оптимальных по Парето x_i точек, удовлетворяющих градиентным Парето-оптимальным условиям:

$$g(\lambda_i, x_i) = (1 - \lambda_i)\cdot f_1'(x_i) + \lambda_i \cdot f_2'(x_i) = 0, \;\; i = 1, \dots, N \qquad (17)$$

Этот набор уравнений недостаточно определен. Для *N* точек имеем *N* уравнений (17) и 2*N* неизвестных x_i и λ_i. Однако можно определить якорные точки x_1 и x_N при $\lambda_1 = 0$ и $\lambda_N = 1$ соответственно: $x_1 \Leftarrow min\big(f_1'(x)\big), \;\; x_N \Leftarrow min\big(f_2'(x)\big)$.

Из условия равномерного распределения точек x_m (*m* = 2, …, *N*-1) между якорными точками x_1 и x_N можно составить *N-2* дополнительных уравнений

$$l_m = \; d_{m-1}^2 - d_{m+1}^2 = f_1(x_m)\cdot\Delta_m(f_1) + f_2(x_m)\cdot\Delta_m(f_2) - \Delta_m^2(f_1) - \; \Delta_m^2(f_2) = 0$$

где: d_{m-1}^2 – квадрат расстояния между x_{m-1} и x_m; d_{m+1}^2 – квадрат расстояния между x_m и x_{m+1};

$\Delta_m(\varphi) = 2\big(\varphi(x_{m+1}) - \varphi(x_{m-1})\big), \;\; \Delta_m^2(\varphi) = \; \varphi^2(x_{m+1}) - \varphi^2(x_{m-1})$.

Тогда имеем связанную систему уравнений ($i = 2, \dots, N-1$):

$$\begin{aligned} &g(\lambda_i, x_i) = 0, \\ &l_i(x_i) = 0 \end{aligned} \qquad (18)$$

Для решения системы (18) используем метод Ньютона. Каждому *(n+1)*-му итерационному шагу этого метода соответствуют следующие соотношения:

$$\begin{aligned} &h_{xi}\cdot\Delta x_i \; + g_{\lambda i}\cdot\Delta\lambda_i \; = -g_i, \\ &l_i'\cdot\Delta x_i = -l_i \end{aligned} \qquad (19)$$

где: $g_i = (1 - \lambda_i)\cdot f_1'(x_i) + \lambda_i \cdot f_2'(x_i)$, $h_{xi} = (1 - \lambda_i)\cdot f_1''(x_i) + \lambda_i \cdot f_2''(x_i)$, $g_{\lambda i} = f_2'(x_i) - f_1'(x_i)$ и
$l_i = l_i(x_i)$, $\quad l_i' = \partial l_i/\partial x_i = f_1'(x_i)\cdot\Delta_i(f_1) + f_2'(x_i)\cdot\Delta_i(f_2)$.
Уравнения (19) можно привести к виду (n = 0, 1, 2, …):

$$\begin{aligned} \Delta\lambda_i^{(n+1)} &= [(h_{xi}\cdot l_i/l_i' - g_i)/g_{\lambda i}\,]^{(n)}, \\ \Delta x_i^{(n+1)} &= -[(\,g_i + g_{\lambda i}\cdot\Delta\lambda_i)/h_{xi}]^{(n)}, \\ \lambda_i^{(n+1)} &= \lambda_i^{(n)} + \Delta\lambda_i^{(n+1)}, \\ x_i^{(n+1)} &= x_i^{(n)} + \Delta x_i^{(n+1)} \end{aligned} \qquad (20)$$

За начальное приближение примем (j = 1, 2, …, N):

$$\lambda_j^{(0)} = (j-1)/(N-1), \quad x_j^{(0)} = \left(1 - \lambda_j^{(0)}\right)\cdot x_1 + \lambda_j^{(0)}\cdot x_N$$

Нетрудно видеть, что на каждом итерационном шаге приходится неоднократно (зависит от N) вычислять значения функций f_1, f_2 и их первой и второй производных, что обуславливает целесообразность использования усечённых гипер-дуальных переменных. Программный код, реализующий этот подход, приведен в Приложении 3 (`pareto(.)`). Рассмотрим, в качестве примера, две целевые функции $f_1(x) = 2x^2 + 1$ и $f_2(x) = (x-2)^2 + 1$ (см. рисунок 1). Численные результаты работы, указанного программного кода при N=7, приведены в таблице 5, а их графическая визуализация на рисунке 2.
Компьютерная реализация усечённых гипер-дуальных чисел была выполнена на языке SWIFT операционной системы macOS (см. Приложение 1). Был создан новый тип данных Thdn (truncated-hyper-dual number) и его расширение, которое включает перезагрузку элементарных функций и операций. Отдельно добавлены процедуры решения некоторых вычислительных задач.
Использование этого типа данных приведено в Приложение 2 и 3.

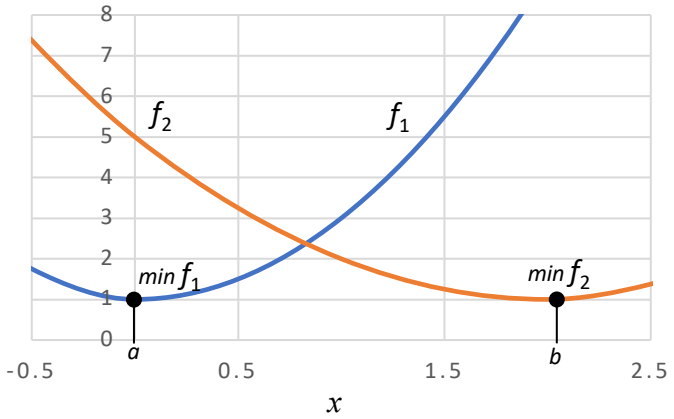

Рисунок 1. Целевые функции: $f_1(x) = 2x^2 + 1$, $f_2(x) = (x-2)^2 + 1$

x_i	λ_i	f_1	f_2	d_m
0.0	0.0	1.0	5.0	
0.45111	0.36809	1.40701	3.39903	1.65189
0.88861	0.61525	2.57926	2.23518	1.65189
1.24345	0.76674	4.09235	1.57236	1.65189
1.53388	0.86810	5.70562	1.21726	1.65189
1.78168	0.94227	7.34879	1.04766	1.65189
2.0	1.0	9.0	1.0	

Таблица 5. Результаты расчета для N=7 и $f_1(x) = 2x^2 + 1,\ \ f_2(x) = (x-2)^2 + 1$

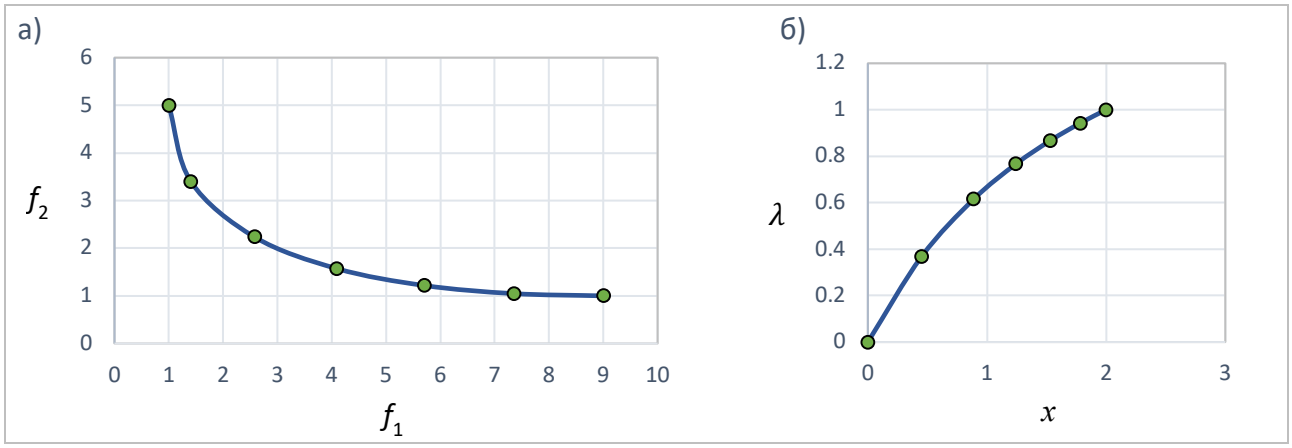

Рисунок 2. а) Фронт Парето (отображение множества Парето $x \in [0;2]$ *на множество значений целевых функций* $f_1 \times f_2$*);*
б) Отображение множества Парето $x \in [0;2]$ *на множество параметра веса* $\lambda \in [0;1]$

Приложение 1

Ниже приведено 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 {...}
    static func g(x:Double, σ:Double, μ:Double) -> Thdn {...}               //--Gaussian function
    static func g(X:Thdn, σ:Double, μ:Double) -> Thdn {...}                 //--Gaussian function
    static func dX(x:Double, y:Double, F:(Thdn,Thdn) -> Thdn) -> Thdn {...}  //--filter
    static func dY(x:Double, y:Double, F:(Thdn,Thdn) -> Thdn) -> Thdn {...}  //--filter
    static func dXY(x:Double, y:Double, F:(Thdn,Thdn) -> Thdn) -> Thdn {...} //--filter
    private static func HD(X:Thdn, f:(Double)-> Thdn) -> {...}
    private static func HD(X:Thdn, f:(Double, Double) -> Thdn, n:Double)-> Thdn {...}
    private static func ZX(F:Thdn,X:Thdn) -> Thdn {...}
    //--SOLUTIONS:
        //--Newton's method for F(x)
    static func newton(x0:Double, F:(Double) -> Thdn, E:Double, maxIteraction:Int=100) -> Double
{...}
       //--Newton-Chebyshev's method for F(x)
    static func newtonEx(x0:Double, F:(Double) -> Thdn, E:Double, maxIteraction:Int=100) -> Double
{...}
       //--Newton's method for min(J(x)), where J(x) = (1 - λ)* F1(x) + λ*F2(x)
    static func newtonMin(x0:Double, λ:Double, F:(Double, Double) -> Thdn, E:Double,
maxIteraction:Int=100)
          ->Double {...}
       //--Newton's method for F(x,y)
    static func newtonXY(x0:Double,y0:Double,F:(Thdn, Thdn) -> Thdn, E:Double,
                         maxIteraction:Int=100)->(x:Double,y:Double){...}
}
```

Приложение 2

Ниже приводятся численные 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

//-- Newton's Methods for F(x):
        func cosX_x3(x: Double)-> Thdn {       //--  cos(X)-X³
            return Thdn.cos(x: x) - Thdn.pow(x: x, n: 3)
        }
        var x = Thdn.newton(x0: 0.5, F: cosX_x3, E: 0.00001) // x = 0.8654740331109566
        x = Thdn.newtonEx(x0: 0.5, F: cosX_x3, E: 0.00001)   // x = 0.8654740331016144

//--Filters:
        func FF(X: Thdn, Y: Thdn)-> Thdn {   //--4X² + 3X² - 4XY + X
            return 4.0*(X*X) + 3.0*(Y*Y) - 4*(X*Y) + X
        }
        var dx  = Thdn.dX( x: 1, y: 1, F: FF)   //  Y = 4 + 5•ε + 8•ω
        var dy  = Thdn.dY( x: 1, y: 1, F: FF)   //  Y = 4 + 2•ε + 6•ω
        var dxy = Thdn.dXY(x: 1, y: 1, F: FF)   //  Y = 4 + 7•ε + 6•ω
        var F   =  dx.re                        //  4
        var Fx  =  dx.im1,                      //  5
            Fxx =  dx.im2                       //  8
        var Fy  =  dy.im1,                      //  2
            Fyy =  dy.im2                       //  6
        var Fxy = (dxy.im2 - dy.im2 - dx.im2)/2 // -4

//--Newton's Method for F(x,y):
       var xy = Thdn.newtonXY(x0: 1, y0: 1, F: FF, E: 0.00000001)  //  x=-0.18750  y=-0.12500
```

Приложение 3

Реализация Парето оптимизации на основе усечённых гипер-дуальных чисел типа Thdn (см. *Приложение 1*)

```
    func f1(x:Double)-> Thdn {return 2*Thdn.pow(x: x, n: 2) + Thdn (re:1)}        //-- f₁ = 2x² +
1
    func f2(x:Double)-> Thdn {return Thdn.pow(x: x - 2, n: 2) + Thdn (re:1)}      //-- f₂ =
(x − 2)² + 1
    let xλ:(x:[Double], λ:[Double]) = pareto(N: 7, f1: f1, f2: f2, E: 0.0000001)  //-- get x and λ
    //--calculation f1ᵢ, f2ᵢ and dᵢ from received xᵢ
    var f1_2:[(f1:Double, f2:Double)]=[], di:[Double]=[]
    for i in 0...xλ.x.count - 1 {f1_2.append((f1(x:xλ.x[i]).re, f2(x:xλ.x[i]).re))}
    for i in 1...xλ.x.count - 1{
         di.append(sqrt(pow(f1_2[i-1].f1 - f1_2[i].f1, 2) + pow(f1_2[i-1].f2 - f1_2[i].f2, 2)))
    }
    //--Pareto optimality
    func pareto(N:Int,f1:@escaping (Double)->Thdn,f2:@escaping (Double)->Thdn,
                E:Double,maxIteraction:Int=100)->(x:[Double], λ:[Double]){
        var x = [Double](repeating: 0, count: N), λ = [Double](repeating: 0, count: N)
        var f1_x, f2_x: Thdn, Gi, Hxi, Gλi, Δx, Δx_max, Δλ:Double, LdLi:(Double,Double), j:Int = 0
        func J(x:Double, λ:Double) -> Thdn { return (1 - λ)*f1(x) + λ*f2(x)}
        func get_L2_dL2i(x:[Double], i:Int)-> (Double,Double){
            func dm(m:Int,  f:(Double)->Thdn)->Double{return 2.0 * (f(x[m + 1]).re - f(x[m -
1]).re)}
            func dm2(m:Int, f:(Double)->Thdn)->Double{return  pow(f(x[m + 1]).re, 2) -
```

```
                pow(f(x[m - 1]).re, 2)}
        var f1x,f2x:Thdn,dm_f1,dm_f2,dm2_f1,dm2_f2:Double;
        let L2, Ldx: Double;
        f1x    = f1(x[i]);                      f2x    = f2(x[i]);
        dm_f1  = dm(m: i,  f: f1);              dm_f2  = dm(m: i,  f: f2);
        dm2_f1 = dm2(m: i, f: f1);              dm2_f2 = dm2(m: i, f: f2);
        L2 = f1x.re * dm_f1 + f2x.re * dm_f2 -  dm2_f1 -  dm2_f2;
        Ldx = f1x.im1 * dm_f1 + f2x.im1 * dm_f2;
        return (L2, Ldx)
    }
    //-- set weights (λ)
    for i in 0...λ.count - 1 {λ[i] = Double(i)/Double(N - 1)}
    //--get anchor points
    x[0]   =  Thdn.newtonMin(x0: 1, λ: 0, F: J, E: E)
    x[N-1] =  Thdn.newtonMin(x0: 1, λ: 1, F: J, E: E)
    //--set intermediate points
    for i in 1...x.count - 2 {x[i] = (1.0 - λ[i]) * x[0] + λ[i] * x[N-1]}
    while true {
        Δx_max = 0.0
        for i in 1...x.count - 2 {
            LdLi = get_L2_dL2i(x: x, i: i)              //-- get l and l'
            f1_x = f1(x[i]);         f2_x = f2(x[i])
            Gi  = (1.0 - λ[i])*f1_x.im1 + λ[i]*f2_x.im1
            Hxi = (1.0 - λ[i])*f1_x.im2 + λ[i]*f2_x.im2
            Gλi = f2_x.im1  - f1_x.im1
            Δλ = (Hxi*LdLi.0/LdLi.1 - Gi)/Gλi           //-- new Δλ
            Δx = -(Gi + Gλi*Δλ)/Hxi                     //-- new Δx
            λ[i] += Δλ
            x[i] += Δx
            Δx_max = max(Δx_max, abs(Δx))
        }
        j += 1
        if Δx_max <= E || j > maxIteraction {break}
    }
    return (x:x, λ:λ)
}
```

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] 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.

[9] А. Н. Колмогоров. О представлении непрерывных функций нескольких переменных в виде суперпозиций непрерывных функций одной переменной и сложения. - ДАН СССР, 1957. – Т. 114, вып. 5. - 953-956 с.

[10] В. А. Даугавет, М. В. Киреева. Приближение функции двух переменных произведением функций одной переменной в заданной области. – Вестник СПбГУ. Сер. 1, 2010, вып.3. 13-21с.

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 и Ньютона-Чебышева на основе усечённых гипер-дуальных функций одной и двух переменных, а также приведено использование усечённых гипер-дуальных функций в процедуре Парето оптимизации. 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, Парето оптимизация*

23 мая 2019 г.