ЧИСЛЕННОЕ РЕШЕНИЕ НЕЛИНЕЙНОЙ ЗАДАЧИ КОШИ 2-ГО ПОРЯДКА С ИСПОЛЬЗОВАНИЕМ УСЕЧЕННЫХ ГИПЕР-ДУАЛЬНЫХ ЧИСЕЛ

Олифер В. И.

1. Постановка задачи и схема ее решения

Рассматривается нормальное нелинейное дифференциальное уравнение 2-го порядка

$$y'' = F(y, y') + \Phi(t), \quad y \equiv y(t), \quad t \in [0,1], \tag{1.1}$$

где: $F(y, y')$ – функция, определяющая нелинейную часть исходного уравнения; $\Phi(t)$ – заданная непрерывно дифференцируемая функция аргумента t; $(\,)' = d(\,)/dt$ – полная производная.

Задача Коши, соответствующая уравнению (1.1), имеет вид

$$\begin{cases} y' = x \\ x' = F(y, y') + \Phi(t) \end{cases} \tag{1.2}$$

с начальными условиями $y(0) = y_0, \;\; x(0) = y'(0) = x_0$.

Для получения численного решения (1.2) (в виде табличных функций) разобьем интервал интегрирования $[0,1]$ отрезками (шагом) малой длины Δt на n частей. В результате получим расчетные точки (узлы) $t_0, t_1, \dots, t_n$, где $t_i = i{\cdot}\Delta t$ и $i = 0, 1, \dots, n$. Пусть в узле t_i решение известно, тогда решение в узле t_{i+1} представим первыми четырьмя членами ряда Тейлора [1]:

$$\begin{aligned} y_{i+1} &= y(t_i + \Delta t) \approx y_i + y_i'{\cdot}\Delta t + \tfrac{1}{2} y_i''{\cdot}\Delta t^2 + \tfrac{1}{6} y_i'''{\cdot}\Delta t^3, \\ x_{i+1} &= x(t_i + \Delta t) \approx x_i + x_i'{\cdot}\Delta t + \tfrac{1}{2} x_i''{\cdot}\Delta t^2 + \tfrac{1}{6} x_i'''{\cdot}\Delta t^3 \;, \end{aligned} \tag{1.3}$$

В точке t_0 решение определяется начальными условиями y_0 и x_0. Для реализации (1.3) помимо значений y_i и x_i необходимо знать величины y_i', y_i'', y_i''', x_i', x_i'' и x_i''', которые можно вычислить по схеме:

$input: y_i, x_i$		
$y_i' = x_i,$	$\longrightarrow$	$x_i' = F(y_i, y_i') + \Phi(t_i),$
$y_i'' = x_i',$	$\longrightarrow$	$x_i'' = F_1(y_i, y_i', y_i'',) + \Phi'(t_i),$
$y_i''' = x_i'',$	$\longrightarrow$	$x_i''' = F_2(y_i, y_i', y_i'', y_i''') + \Phi''(t_i),$
		$output: y_{i+1}, \; x_{i+1}$ см. (1.3)

(1.4)

Таблица 1.1. Схема расчета i-го узла

где:

$F(y_i, y_i') = F(y_i, x_i)$,
$F_1(y_i, x_i, x_i') = F'(y_i, x_i) = F_y \cdot x_i + F_x \cdot x_i'$,
$F_2(y_i, x_i, x_i', x_i'') = F''(y_i, x_i) = F_1'(y_i, x_i, x_i') = F_y' \cdot x_i + F_x' \cdot x_i' + F_{x'}' \cdot x_i'' =$
$= F_{yy} \cdot x_i^2 + 2F_{xy} \cdot x_i \cdot x_i' + F_{xx}(x_i')^2 + F_y \cdot x_i' + F_x \cdot x_i''$
и

$$(\ )_y = \frac{\partial(\ )}{\partial y},\ (\ )_x = \frac{\partial(\ )}{\partial x},\ (\ )_{yx} = \frac{\partial^2(\ )}{\partial y \partial x},\ (\ )_{yy} = \frac{\partial^2(\ )}{\partial y^2}, \qquad (\ )_{xx} = \frac{\partial^2(\ )}{\partial x^2}.$$

Если $F(y, y')$ представима в виде $F(y, y') = F(y, x) = f_1(y) + f_2(x) + f_3(y) \cdot f_4(x)$, что возможно в большинстве случаев, то

$F_y(y_i, x_i) = f_{1,y}(y_i) + f_{3,y}(y_i) \cdot f_4(x_i)$,
$F_x(y_i, x_i) = f_{2,x}(x_i) + f_{4,x}(x_i) \cdot f_3(y_i)$,
$F_{xy}(y_i, x_i) = f_{3,y}(y_i) \cdot f_{4,x}(x_i)$,
$F_{yy}(y_i, x_i) = f_{1,yy}(y_i) + f_{3,yy}(y_i) \cdot f_4(x_i)$,
$F_{xx}(y_i, x_i) = f_{2,xx}(x_i) + f_{4,xx}(x_i) \cdot f_3(y_i)$.

Тогда необходимо вычислять величины восьми производных (первых и вторых производных функций $f_1(y)$, $f_2(x)$, $f_3(y)$, $f_4(x)$), а именно $f_{1,y}(y_i)$, $f_{2,x}(x_i)$, $f_{3,y}(y_i)$, $f_{4,x}(x_i)$, $f_{1,yy}(y_i)$, $f_{2,xx}(x_i)$, $f_{3,yy}(y_i)$ и $f_{4,xx}(x_i)$.
Для многих практически важных задач можно положить $f_1(y) = \alpha_0 \cdot y + \alpha_1 \cdot y^2$, $f_2(x) = \alpha_2 \cdot x + \alpha_3 \cdot (x)^2$, $f_3(y) = \alpha_4 \cdot y$, $f_4(x) = x$.
В этом случае:
$F(y, x) = \alpha_0 \cdot y + \alpha_1 \cdot y^2 + \alpha_2 \cdot x + \alpha_3 \cdot (x)^2 + \alpha_4 \cdot y \cdot x$,
$F_y(y, x) = \alpha_0 + 2\alpha_1 y + \alpha_4 x$, $F_x(y, x) = \alpha_2 + 2\alpha_3 x + \alpha_4 y$,
$F_{xy}(y, x) = \alpha_4$, $F_{yy}(y, x) = 2\alpha_1$, $F_{xx}(y, x) = 2\alpha_3$
и
$F_1(y, x, x') = \alpha_0 \cdot x + 2\alpha_1 \cdot y \cdot x + \alpha_2 \cdot x' + 2\alpha_3 \cdot x \cdot x' + \alpha_4 (x^2 + y \cdot x')$,
$F_2(y, x, x', x'') = \alpha_0 \cdot x' + 2\alpha_1 (x^2 + y \cdot x') + \alpha_2 \cdot x'' + 2\alpha_3 ((x')^2 + x \cdot x'') +$
$+ \alpha_4 (3x \cdot x' + y \cdot x'')$.
Как видно из изложенного выше, использование метода разложения в ряд Тейлора требует вычисление производных различных порядков. Если исходные функции достаточно сложные или более того заданы композитным программным кодом, то вычисление значений их производных становится весьма трудоемкой, а иногда и невыполнимой задачей, что является главным недостатком метода разложения в ряд Тейлора. Однако с развитием вычислительной техники возникло новое направление в вопросах численного дифференцирования – автоматическое дифференцирование [2, 3], связанное с точным (точностью представления чисел в компьютерной системе) вычислением производных сложных математических функций. Автоматическое дифференцирование (АД) позволяет избежать дублирование функциональности программного кода (изменение кода функции не требует изменения кода ее производной). Для компьютерной реализации АД необходимо

создать новый тип данных, перезагрузить базовые математические функции и операции над ними. Если новый тип данных строится на основе гипер-дуальных чисел [4, 5] или усеченных гипер-дуальных чисел [6], то за одно обращение к перезагруженной функции точно вычисляются значения самой функции и ее первой и второй производных.

2. Усеченные гипер-дуальные числа

Впервые усеченные гипер-дуальные числа были описаны в работе [6]. Согласно которой усеченное гипер-дуальное число (truncated hyper-dual number) определяется выражением $X = x + x_1\boldsymbol{\varepsilon} + x_2\boldsymbol{\omega}$, где x, x_1 и x_2 – вещественные числа, $\boldsymbol{\varepsilon}$ и $\boldsymbol{\omega}$ – мнимые символы. Пространство усеченных гипер-дуальных чисел отвечает трехмерной алгебре с правилом умножения элементов базиса $\{\mathbf{1},\ \boldsymbol{\varepsilon},\ \boldsymbol{\omega}\}$:

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

(2.1)

Таблица 2.1. Правила умножения элементов базиса усечённых гипер-дуальных чисел

Число $x = Re(X)$ называется главной частью X, а $x_1 = Im_1(X)$ и $x_2 = Im_2(X)$ – мнимыми частями X.

Алгебраические операции сложения, умножения, обращения и деления (с учетом табл. 2.1) определены по правилам:

$$
\begin{aligned}
&A = a + a_1\boldsymbol{\varepsilon} + a_2\boldsymbol{\omega}, \qquad B = b + b_1\boldsymbol{\varepsilon} + b_2\boldsymbol{\omega}\\
&A + B = a + b + (a_1 + b_1)\boldsymbol{\varepsilon} + (a_2 + b_2)\boldsymbol{\omega},\\
&A{\cdot}B = a{\cdot}b + (a{\cdot}b_1 + b{\cdot}a_1)\boldsymbol{\varepsilon} + (a{\cdot}b_2 + 2{\cdot}a_1{\cdot}b_1 + b{\cdot}a_2)\boldsymbol{\omega},\\
&A^{-1} = a^{-1} - a_1{\cdot}a^{-2}\boldsymbol{\varepsilon} + (2{\cdot}a_1^2{\cdot}a^{-3} - a_2{\cdot}a^{-2})\boldsymbol{\omega},\\
&{}^{A}/_{B} = A{\cdot}B^{-1} = a{\cdot}b^{-1} + (a_1{\cdot}b^{-1} - a{\cdot}b_1{\cdot}b^{-2})\boldsymbol{\varepsilon} + [2{\cdot}(a{\cdot}b_1^2{\cdot}b^{-3} - a_1{\cdot}b_1{\cdot}b^{-2}) -\\
&\qquad\qquad a{\cdot}b_2{\cdot}b^{-2} + a_2{\cdot}b^{-1}]\boldsymbol{\omega}
\end{aligned}
\tag{2.2}
$$

Функция усеченного гипер-дуального аргумента реализуется выражением

$$F(X) = f(x) + x_1{\cdot}f'(x)\boldsymbol{\varepsilon} + (x_2{\cdot}f'(x) + x_1^2{\cdot}f''(x))\boldsymbol{\omega} \tag{2.3}$$

При $x_1 = 1$ и $x_2 = 0$ выражение (2.3) принимает вид:

$$F(X) = f(x) + f'(x)\boldsymbol{\varepsilon} + f''(x)\boldsymbol{\omega} \tag{2.4}$$

Описание элементарных (базовых) функций усечённого гипер-дуального аргумента приведены в [6]. Например, $\ln(X) = \ln(x) + x^{-1}\boldsymbol{\varepsilon} - x^{-2}\boldsymbol{\omega}$, откуда $Re(X) = f(x) = \ln(x)$, $Im_1(X) = f'(x) = x^{-1}$ и $Im_2(X) = f''(x) = -x^{-2}$.

Вычисление сложной усеченной гипер-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 $F_{n-1} = f_{n-1}(F_n, X)$, $F_{n-2} = f_{n-2}(F_{n-1}, X), \dots, F = f_1(F_2, X)$.

3. Компьютерная реализация

Использую программный код на языке SWIFT 5, описывающий новый тип данных `Thdn` (truncated hyper-dual number) [6], были составлены две процедуры `Cauchy_Taylor()` и `Cauchy_TaylorEx()` (см. Приложение 1) реализующих решение задачи Коши (1.2). Процедура `Cauchy_Taylor()` обеспечивает решение поставленной задачи в случае когда нелинейная часть исходного уравнения представима в виде $F(y,x) = \alpha_0 \cdot y + \alpha_1 \cdot y^2 + \alpha_2 \cdot x + \alpha_3 \cdot (x)^2 + \alpha_4 \cdot y \cdot x$. Для более общего случая $F(y,x) = f_1(y) + f_2(x) + f_3(y) \cdot f_4(x)$ предназначена процедура `Cauchy_TaylorEx()`. Примеры применения указанных процедур даны в Приложение 2.

4. Численный эксперимент

Для оценки точности результатов, получаемых по процедурам `Cauchy_Taylor()` и `Cauchy_TaylorEx()` (см. Приложение 1 и 2), была выполнена серия расчетов следующих задач Коши:

1. $y''(t) = 2 \cdot y'(t) + 3 \cdot y(t) + e^{4t}$, $y(0) = y_0 = 2.2$, $x(0) = x_0 = 2.8$, $n = 10$; точное решение $y(t) = e^{-t} + e^{3t} + 0.2 \cdot e^{4t}$.
2. $y''(t) = 0.5 \cdot \left(y^{-1}(t) \cdot \left(y'(t)\right)^2 - y^{-1}(t)\right)$, $y(0) = y_0 = 5/12$, $x(0) = x_0 = 1.5$, $n = 10$; точное решение $y(t) = 3 \cdot (t+1)^2/4 \ -1/3$.
3. $y''(t) = -0.2y'(t) - 10\sin\left(y(t)\right)$, $t \in [0, 0.5]$, $y(0) = 0.3$, $y'(0) = 0$, $n = 5, 10, 20$ – уравнение колебания в сопротивляющейся среде. Точное решение этой задачи Коши выражается через функции Бесселя первого рода и в точке $t = 0.5$ дает $y(0.5) = 0.0088846$.

Полученные результаты расчетов приведены в табл. 4.1, 4.2 и 4.3. Нетрудно видеть, что процедуры `Cauchy_Taylor()` и `Cauchy_TaylorEx()` дают одинаковые результаты. Для линейной задачи (1) уже при $n = 10$ максимальное относительное отклонение в точке $t = 1$ от точного решения составило ~ 0.0034. Для нелинейной задачи (2) при $n = 10$ результаты работы процедуры `Cauchy_TaylorEx()` полностью совпадают с точным решением. Процедура `Cauchy_Taylor()` не может быть применена к задаче (2) т.к. `Cauchy_Taylor()` не предполагает наличие обратных величин в правой части уравнения. Следует заметить, что численное решение задачи (2) при удержании в (1.3) первых трех слагаемых дает практически точный результат, а при удержании только двух членов – относительная погрешность составляет

~0.02. Это объясняется сравнительной простатой задачи (2) точное решение которой является алгебраическим многочленом второго порядка. Для задачи (3) относительные погрешности решения при $n = 5,\ 10,\ 20$ составляют $\sim 1.3 \cdot 10^{-3}$, $\sim 6.5 \cdot 10^{-4}$ и $\sim 2.3 \cdot 10^{-5}$ соответсвенно.

t	Точное решение		`Cauchy_Taylor()`		`Cauchy_TaylorEx()`	
	y	$x = y'$	y	$x = y'$	y	$x = y'$
0.0	2.2	2.8	2.2	2.8	2.2	2.8
0.1	2.553061	4.338199	2.552467	4.336200	2.552467	4.336200
0.2	3.085958	6.4280584	3.084298	6.422554	3.084298	6.422554
0.3	3.864445	9.294085	3.860974	9.282695	3.860974	9.282695
0.4	4.981043	13.252457	4.974596	13.231474	4.974596	13.231474
0.5	6.566031	18.749781	6.554807	18.713488	6.554807	18.713488
0.6	8.803094	26.418672	8.784334	26.358318	8.784334	26.358318
0.7	11.951685	37.157642	11.921189	37.059926	11.921189	37.059926
0.8	16.379011	52.246224	16.330424	52.091026	16.330424	52.091026
0.9	22.605948	73.511213	22.529692	73.268232	22.529692	73.268232
1.0	31.373046	103.567251	31.254742	103.191006	31.254742	103.191006

Таблица 4.1. Результаты расчета задачи Коши $y''(t) = 2y'(t) + 3y(t) + e^{4t}$

t	Точное решение		`Cauchy_TaylorEx()`	
	y	$x = y'$	y	$x = y'$
0.0	0.416667	1.500000	0.416667	1.500000
0.1	0.574167	1.650000	0.574167	1.650000
0.2	0.746667	1.800000	0.746667	1.800000
0.3	0.934167	1.950000	0.934167	1.950000
0.4	1.136667	2.100000	1.136667	2.100000
0.5	1.354167	2.250000	1.354167	2.250000
0.6	1.586667	2.400000	1.586667	2.400000
0.7	1.834167	2.550000	1.834167	2.550000
0.8	2.096667	2.700000	2.096667	2.700000
0.9	2.374167	2.850000	2.374167	2.850000
1.0	2.666667	3.000000	2.666667	3.000000

Таблица 4.2. Результаты расчета задачи Коши $y''(t) = 0.5 \cdot \left(y^{-1}(t) \cdot \left(y'(t) \right)^2 - y^{-1}(t) \right)$

t	$n = 5$		$n = 10$		$n = 20$	
	y	$x = y'$	y	$x = y'$	y	$x = y'$
0.0	0.3000000	0.0000000	0.3000000	0.0000000	0.3000000	0.0000000
0.1	0.2853225	-0.2878794	0.2854244	-0.2879293	0.2854367	-0.2879401

0.2	0.2433014	-0.5426818	0.2434868	-0.5429459	0.2435084	-0.5429873
0.3	0.1784850	-0.7404521	0.1787127	-0.7410791	0.1787377	-0.7411678
0.4	0.0975720	-0.8625426	0.0977799	-0.8636338	0.0978000	-0.8995875
0.5	0.0087669	-0.8978263	0.0088788	-0.8993888	0.0088844	-0.8995875

Таблица 4.3. Результаты расчета задачи Коши $y''(t) = -0.2y'(t) - 10\,sin(y(t))$ *при разном количестве расчетных точек*

Как видно, повысить точность расчета можно за счет увеличения количества расчетных точек. Однако возможен и другой подход, заключающийся в удержании пяти членов разложений Тейлора (1.3), т.е. учете четвертых производных. В этом случае следует использовать супер-дуальные числа третьего класс [7].

Приложение 1

Ниже приведен код функций `Cauchy_TaylorEx()` и `Cauchy_Taylor()` для Swift 5 (macOS 10.14). Тип данных `Thdn` дан в [6].

```
//-------------------------------------------------------------
//          Φ      - predefined function
//          f1(y), f2(x), f3(y), f4(x) - predefined functions
//          y0, x0 - initial conditions
//          t0, tn - start and end of integration  interval
//          n      - number of parts into which the interval [t0, tn] is divided

func Cauchy_TaylorEx(Φ:(Double)->Thdn,f1:(Double)->Thdn,f2:(Double)->Thdn,
                                        f3:(Double)->Thdn,f4:(Double)->Thdn,
                     y0:Double,  x0:Double, t0:Double, tn:Double,
                     n:Int)->[(t:Double, y:Double, x:Double)]{
    let Δt:Double = (tn - t0)/Double(n);
    func  yx(z:(_:Double,_:Double,_:Double,_:Double))->Double{
        return  z.0 + Δt*z.1 + Δt*Δt*z.2/2.0 + Δt*Δt*Δt*z.3/6.0;
    }
    var Yi:(y:Double,y1:Double,y2:Double,y3:Double) = (y:y0,y1:0,y2:0,y3:0);
    var Xi:(x:Double,x1:Double,x2:Double,x3:Double) = (x:x0,x1:0,x2:0,x3:0);
    var tXY:[(t:Double, y:Double, x:Double)] = [];
    var φ:Thdn;
    var r1,r2,r3,r4:Thdn;
    var F, F1, F2, Fy, Fx, Fxy, Fyy, Fxx:Double;
    tXY.append((t:t0, y:y0, x:x0));
    for t in stride(from: t0, to: tn, by: Δt) {
        r1 = f1(Yi.y);            r2 = f2(Xi.x);
        r3 = f3(Yi.y);            r4 = f4(Xi.x);
        F   = r1.re + r2.re + r3.re*r4.re;
        Fy  = r1.im1 + r3.im1*r4.re;     Fx  = r2.im1 + r4.im1*r3.re;
        Fxy = r3.im1*r4.im1;
        Fyy = r1.im2 + r3.im2*r4.re;     Fxx = r2.im2 + r4.im2*r3.re;
        φ = Φ(t);
        Yi.y1 = Xi.x;        Xi.x1 = F  + φ.re;
        F1 = Fy*Xi.x + Fx*Xi.x1;
        Yi.y2 = Xi.x1;       Xi.x2 = F1 + φ.im1;
        F2 = Fyy*Xi.x*Xi.x + 2.0*Fxy*Xi.x*Xi.x1 + Fxx*Xi.x1*Xi.x1 +
             Fy*Xi.x1 + Fx*Xi.x2;
        Yi.y3 = Xi.x2;       Xi.x3 = F2 + φ.im2;
```

```
        Yi.y  = yx(z: Yi);   Xi.x  = yx(z: Xi);
        tXY.append((t:t + Δt, y:Yi.y, x:Xi.x));
    }
    return tXY;
}

//-------------------------------------------------------------
//          Φ      - predefined function
//          y0, x0 - initial conditions
//          t0, tn - start and end of integration interval
//          n      - number of parts into which the interval [t0, tn] is divided
//          α      - array of numbers

func Cauchy_Taylor(Φ:(Double)->Thdn,
                   y0:Double,  x0:Double, t0:Double, tn:Double,
                   n:Int, α:[Double])->[(t:Double, y:Double, x:Double)]{
    let Δt:Double = (tn - t0)/Double(n);
    func F (y:Double, y1:Double, α:[Double])->Double{
        return α[0]*y + α[1]*y*y + α[2]*y1 + α[3]*y1*y1 + α[4]*y*y1;
    }
    func F1(y:Double, y1:Double, y2:Double, α:[Double])->Double{
        return α[0]*y1 + 2.0*α[1]*y*y1 + α[2]*y2 + 2.0*α[3]*y1*y2 + α[4]*(y1*y1 +
               y*y2);
    }
    func F2(y:Double, y1:Double, y2:Double, y3:Double, α:[Double])->Double{
        return α[0]*y2 + 2.0*α[1]*(y1*y1 + y*y2) + α[2]*y3 +
               2.0*α[3]*(y2*y2 + y1*y3) + α[4]*(3.0*y1*y2 + y*y3);
    }
    func yx(z:(_:Double,_:Double,_:Double,_:Double))->Double {
         return  z.0 + Δt*z.1 + Δt*Δt*z.2/2.0 + Δt*Δt*Δt*z.3/6.0;
    }
    var Yi:(y:Double,y1:Double,y2:Double,y3:Double) = (y:y0,y1:0,y2:0,y3:0);
    var Xi:(x:Double,x1:Double,x2:Double,x3:Double) = (x:x0,x1:0,x2:0,x3:0);
    var tXY:[(t:Double, y:Double, x:Double)] = [];
    var φ:Thdn;
    tXY.append((t:t0, y:y0, x:x0));
    for t in stride(from: t0, to: tn, by: Δt) {
        φ = Φ(t);
        Yi.y1 = Xi.x;         Xi.x1 = F(y: Yi.y, y1: Yi.y1, α: α) + φ.re;
        Yi.y2 = Xi.x1;        Xi.x2 = F1(y: Yi.y, y1: Yi.y1, y2: Yi.y2, α: α) + φ.im1;
        Yi.y3 = Xi.x2;        Xi.x3 = F2(y: Yi.y, y1: Yi.y1, y2: Yi.y2, y3: Yi.y3,
                                    α: α) + φ.im2;
        Yi.y  = yx(z: Yi);    Xi.x  = yx(z: Xi);
        tXY.append((t:t + Δt, y:Yi.y, x:Xi.x));
    }
    return tXY;
}
```

Приложение 2

Код для численного эксперимента на языке Swift 5 (macOS 10.14). Тип данных `Thdn` приведен в [6].

```
//  TASK (1): y''(t) = 2y'(t) + 3y(t) + e^{4t},  y(0) = y_0 = 2.2,  x(0) = x_0 = 2.8

func Φ(t:Double)->Thdn{return  Thdn.exp(X:4.0*Thdn(re: t, im1: 1, im2: 0))}
func f1(y:Double)->Thdn{return 3.0*Thdn(re: y, im1: 1, im2: 0)}
func f2(x:Double)->Thdn{return 2.0*Thdn(re: x, im1: 1, im2: 0)}
func f3(y:Double)->Thdn{return Thdn(re: 0)}
```

```
func f4(x:Double)->Thdn{return Thdn(re: 0)}


let ct_ex1 = Cauchy_TaylorEx(Φ: Φ, f1: f1, f2: f2, f3: f3, f4: f4, y0: 2.2, x0: 2.8,
                             t0: 0.0, tn: 1.0, n: 10);

let ct = Cauchy_Taylor(Φ: Φ, y0: 2.2, x0: 2.8, t0: 0.0, tn: 1.0, n: 10,
                       α: [3.0, 0.0, 2.0, 0.0, 0.0]);

//  TASK (2): y''(t) = 0.5·(y^-1(t)·(y'(t))^2 − y^-1(t)),  y(0) = y_0 = 5/12, x(0) = x_0 = 1.5

func Φ0 (t:Double)->Thdn{return Thdn(re: 0)}
func f11(y:Double)->Thdn{return Thdn(re: -0.5)/Thdn(re: y, im1: 1, im2: 0)}
func f21(x:Double)->Thdn{return Thdn(re: 0.0)}
func f31(y:Double)->Thdn{return Thdn (re: 0.5)/Thdn(re: y, im1: 1, im2: 0)}
func f41(x:Double)->Thdn{return Thdn.pow(x: x, n: 2)}

let ct_ex2 = Cauchy_TaylorEx(Φ: Φ0, f1: f11, f2: f21, f3: f31, f4: f41, y0: 5.0/12.0,
x0: 1.5, t0: 0.0, tn: 1.0, n: 10);

//  TASK (3): y''(t) = −0.2y'(t) − 10 sin(y(t)),  y(0) = y_0 = 0.3, x(0) = x_0 = 0.0

func f12(y:Double)->Thdn{return Thdn(re: -10.0)*Thdn.sin(X:Thdn(re:y, im1:1, im2:0))}
func f22(x:Double)->Thdn{return Thdn(re: -0.2)*Thdn(re: x, im1: 1, im2: 0)}
func f32(y:Double)->Thdn{return Thdn(re: 0.0)}
func f42(x:Double)->Thdn{return Thdn(re: 0.0)}

let ct_ex3 = Cauchy_TaylorEx(Φ: Φ0, f1: f12, f2: f22, f3: f32, f4: f42,
                             y0: 0.3, x0: 0.0, t0: 0.0, tn: 0.5, n: 10);
```

Литература

1. *Авдюшев В. А.* Численные методы интегрирования обыкновенных дифференциальных уравнений. Томск: Изд-во ТГУ, 2009, 52с.
2. *Naumann U.* The art of differentiating computer programs. Society for industrial and applied mathematics, Philadelphia, USA, 2012.
3. *Corliss G., Faure C., Griewank A., Hascolt L., Naumann U.* Automatic Differentiation Bibliography // Automatic Differentiation of Algorithms: From Simulation to Optimization. Springer, 2002. p. 383—425.
4. *Fike J.A., Alonso J.J.* The development of hyper-dual numbers for exact second derivative calculations. AIAA paper 2011-886, 49th AIAA Aerospace Sciences Meeting, January 4-7, 2011.
5. *Fike J.A.* Multi-objective optimization using hyper-dual numbers. Ph.D. Dissertation, Stanford University, 2013.
6. *Олифер В. И.* Усеченные гипер-дуальные числа в автоматическом дифференцировании. –URL: http://viosolutions.amerihomesrealty.com/pdf/Усеченные_гипер-дуальные_числа_в_автоматическом_дифференцировании.pdf (дата обращения: 01.04.2020).
7. *Олифер В. И.* Автоматическое дифференцирование на основе супер-дуальных чисел. –

URL: http://viosolutions.amerihomesrealty.com/pdf/Автоматическое_дифференцирование_на_основе_супер-дуальных_чисел.pdf
(дата обращения: 01.04.2020).

Абстракт

В данной публикации рассматривается численный метод решения нелинейной задачи Коши второго порядка, основанный на использовании разложения в ряд Тейлора и автоматического дифференцирования на базе специальных дуальных чисел (усеченных гипер-дуальных чисел). Представлена компьютерная реализация этого метода для языка SWIFT операционной системы macOS на основе которой проведены численные эксперименты.

Ключевые слова*: нелинейная задача Коши, усеченные гипер-дуальные числа, автоматическое дифференцирование, Cauchy problem, truncated hyper-dual numbers, automatic differentiation*

---

20 апреля 2020 г.