К ОПРЕДЕЛЕНИЮ ДЛИНЫ ДУГИ И ПЛОЩАДИ ПОВЕРХНОСТИ ВРАЩЕНИЯ НА ОСНОВЕ ГИПЕР-ДУАЛЬНЫХ ЧИСЕЛ

Олифер В.И.

В интегральном исчислении эллиптический интеграл появился в связи с задачей вычисления длины дуги эллипса и был впервые исследован Джулио Фаньяно в первой половине XVIII века, а позднее –Леонардом Эйлером. Вычисление длины дуги и площади поверхности вращения определяются следующими эллиптическими интегралами

$$L = \int_a^b \sqrt{1 + \dot{y}^2}\, dx, \qquad S = 2\pi \int_a^b y\sqrt{1 + \dot{y}^2}\, dx \tag{1}$$

соответственно, которые относятся к эллиптическим интегралам 2-го рода [1, 7]. Функция $y(x)$, должна быть непрерывной на отрезке $[a, b]$ вместе со своей первой производной $\dot{y}(x) = dy/dx$.

В общем случае эти эллиптические интегралы не выражаются через элементарные функции. По определению, элементарные функции – функции, определяемые формулами, содержащими конечное число алгебраических или тригонометрических операций, производимых над аргументом, функцией и некоторыми постоянными. Поэтому на практике используют различные процедуры аппроксимации.

Здесь же рассматривается метод интегрирования выражений (1), основанный на автоматическом дифференцировании [5, 6] с использованием усеченных гипре-дуальных чисел [2].

Рассмотрим функцию $f(x)$, непрерывную на отрезке $[a, b]$ вместе со своей первой и второй производной. Формулу численного интегрирования функции $f(x)$, приведенную в [3], можно преобразовать к виду

$$\int_a^b f(x)\, dx \approx \frac{\Delta x}{4}\Bigg\{f_{0n} + \ p_1(f_{11} + \ f_{2n}) + \ p_2\big(\dot{f}_{11} - \dot{f}_{2n}\big) + \ p_3\big(\ddot{f}_{11} + \ddot{f}_{2n}\big) + \sum_{n=1}^{n-1}\big[f_{0i} + 2(\,p_1 f_{2i} + \ p_3 \ddot{f}_{2i})\big]\Bigg\} \tag{2}$$

где:

$$\Delta x = \frac{a-b}{n}, \quad a > b, \quad x_i \in [a,b], \quad x_0 = a, \quad x_n = b,$$

$$f_{1i} = f(x_i), \quad f_{2i} = f(x_i + \Delta x), \quad f_{0i} = f(x_i - \Delta x/2), \quad \dot{f} = \frac{df(x)}{dx}, \quad \ddot{f} = \frac{d^2 f(x)}{dx^2},$$

$$p_1 = \frac{3}{2}, \quad p_2 = \Delta x\,/4\,, \quad p_3 = p_2^2/3$$

В соотношении (2) изначально предполагалось, что на каждом интервале $[x_i, x_{i+1}]$ исходная функция $f(x)$ аппроксимируется многочленом третьей степени относительно точки $(x_i + x_{i+1})/2$ и введением корректирующей функции, использующей функцию Хэвисайда [4].

Как видно, для реализации (2) требуется вычисление значений исходной функции и ее первой и второй производных в расчетных точках. Далее для их нахождения будем использовать метод автоматического дифференцирования, основанный на использовании усеченных гипер-дуальных чисел [2].

Автоматическое дифференцирование [5, 6] связанно с точным (точностью представления чисел в компьютерной системе) вычислением производных сложных математических функций. Автоматическое дифференцирование (АД) позволяет избежать дублирования 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 усеченных гипер-дуальных чисел [2], то за одно обращение к перезагруженной функции точно вычисляются значения самой функции и ее первой и второй производных.

Согласно [2] 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

(3)

Таблица 1. Правила умножения элементов базиса усечённых гипер-дуальных чисел.

Число $x = Re(X) = X.Re$ называется главной частью X, а $x_1 = Im_1(X) = X.Im1$ и $x_2 = Im_2(X) = X.Im2$ – мнимыми частями X.

Алгебраические операции сложения, умножения, обращения и деления (с учетом табл. 1) определены по правилам:

$$\begin{aligned}
&A = a + a_1\boldsymbol{\varepsilon} + a_2\boldsymbol{\omega}, \qquad B = b + b_1\boldsymbol{\varepsilon} + b_2\boldsymbol{\omega}\\
&A + B = a + b + (a_1 + b_1)\boldsymbol{\varepsilon} + (a_2 + b_2)\boldsymbol{\omega},\\
&A{\cdot}B = a{\cdot}b + (a{\cdot}b_1 + b{\cdot}a_1)\boldsymbol{\varepsilon} + (a{\cdot}b_2 + 2{\cdot}a_1{\cdot}b_1 + b{\cdot}a_2)\boldsymbol{\omega},\\
&A^{-1} = a^{-1} - a_1{\cdot}a^{-2}\boldsymbol{\varepsilon} + (2{\cdot}a_1^2{\cdot}a^{-3} - a_2{\cdot}a^{-2})\boldsymbol{\omega},\\
&{}^{A}/_{B} = A{\cdot}B^{-1} = a{\cdot}b^{-1} + (a_1{\cdot}b^{-1} - a{\cdot}b_1{\cdot}b^{-2})\boldsymbol{\varepsilon} + [2{\cdot}(a{\cdot}b_1^2{\cdot}b^{-3} - a_1{\cdot}b_1{\cdot}b^{-2}) -\\
&\qquad\qquad a{\cdot}b_2{\cdot}b^{-2} + a_2{\cdot}b^{-1}]\boldsymbol{\omega}
\end{aligned} \tag{4}$$

Функция усеченного гипер-дуального аргумента реализуется выражением

$$F(X) = f(x) + x_1\dot{f}(x)\boldsymbol{\varepsilon} + (x_2\dot{f}(x) + x_1^2\ddot{f}(x))\boldsymbol{\omega} \tag{5}$$

При $x_1 = 1$ и $x_2 = 0$ выражение (5) принимает вид:

$$F(X) = f(x) + \dot{f}(x)\boldsymbol{\varepsilon} + \ddot{f}(x)\boldsymbol{\omega} \tag{6}$$

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

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 $F_{n-1} = f_{n-1}(F_n, X)$, $F_{n-2} = f_{n-2}(F_{n-1}, X)$, … , $F = f_1(F_2, X)$.

Для применения АД (на базе усеченных гипер-дуальных чисел и усеченных гипер-дуальных функций) к соотношению (2) необходимо выполнить следующие отображения

$$x \to X = x + 1{\cdot}\boldsymbol{\varepsilon} + 0{\cdot}\boldsymbol{\omega}, \quad f(x) \to F(X) = f(x) + \dot{f}(x){\cdot}\boldsymbol{\varepsilon} + \ddot{f}(x){\cdot}\boldsymbol{\omega} \tag{7}$$

Тогда для (2) имеем

$$\begin{aligned}
&f_{1i} = F(X_i).Re, \quad f_{2i} = F(X_i + \bar{X}_i).Re, \quad f_{0i} = F(X_i - 0.5\bar{X}_i).Re,\\
&\dot{f}_{1i} = F(X_{1i}).Im1, \quad \ddot{f}_{1i} = F(X_{1i}).Im2, \quad \dot{f}_{2i} = F(X_{2i}).Im1, \quad \ddot{f}_{2i} = F(X_{2i}).Im2,\\
&X_{1i} = x_{1i} + 1{\cdot}\boldsymbol{\varepsilon} + 0{\cdot}\boldsymbol{\omega}, \quad X_{2i} = x_{2i} + 1{\cdot}\boldsymbol{\varepsilon} + 0{\cdot}\boldsymbol{\omega}, \quad \bar{X}_i = \Delta x + 1{\cdot}\boldsymbol{\varepsilon} + 0{\cdot}\boldsymbol{\omega}
\end{aligned} \tag{8}$$

Применим эти результаты для вычисления интегралов (1):

$$y(x) \to Y(X) = y(x) + \dot{y}(x)\cdot\boldsymbol{\varepsilon} + \ \ddot{y}(x),$$

для первого интеграла из (1):

$$F_1(Y(X)) = \sqrt{(1 + 0\cdot\boldsymbol{\varepsilon} + \ 0\cdot\boldsymbol{\omega}) + [Y(X).Im1 + Y(X).Im2\cdot\boldsymbol{\varepsilon} + \ 0\cdot\boldsymbol{\omega}]^2} \qquad (9)$$

для второго интеграла из (1):

$$F_2(Y(X)) = Y(X)\cdot\sqrt{(1 + 0\cdot\boldsymbol{\varepsilon} + \ 0\cdot\boldsymbol{\omega}) + [Y(X).Im1 + Y(X).Im2\cdot\boldsymbol{\varepsilon} + \ 0\cdot\boldsymbol{\omega}]^2} \qquad (10)$$

В формулах (9) и (10) операция извлечения квадратного корня выполняется по правилам усеченных гипер-дуальных чисел: $\sqrt{X} = \sqrt{x} + X.Im1\cdot\dot{\sqrt{x}}\,\boldsymbol{\varepsilon} + (X.Im2\cdot\dot{\sqrt{x}} + \ X.Im1^2\cdot\ddot{\sqrt{x}})\boldsymbol{\omega}$.

Используя $F_1(Y(X))$ и $F_2(Y(X))$, как входную информацию для (8), уже можно воспользоваться формулой (2) для нахождения значений исходных интегралов.

Компьютерная реализация усеченных гипер-дуальных чисел была выполнена на языке SWIFT операционной системы macOS. Был создан новый тип данных Thdn (truncated hyper-dual number) и его расширение, которое включает перезагрузку элементарных функций и операций (см. Приложение 1). Отдельно (Приложение 2) представлены процедуры integral(...), LengthCurve(...), SurfaceArea(...) реализующие формулы (2), (9) и (10) соответственно. В Приложении 3 даны примеры обращения к процедурам LengthCurve(...) и SurfaceArea(...).

Следует отметить, что при описании исходной функции $Y(X)$ следует предусмотреть обработку её разрывов в интервале интегрирования. В ряде случаев проблема разрыва исходной функции может быть решена, если последняя имеет параметрическое представление $x = x(t),\ y = y(t)$. При этом длина дуги этой функции определяется соотношением

$$L = \int_{\alpha}^{\beta} \sqrt{\dot{x}^2(t) + \dot{y}^2(t)}\, dt \qquad (11)$$

Например, для уравнения эллипса ($a,\ b$ - полуоси эллипса) $y = b\sqrt{1 - x^2/a^2}$ видно, что первая производная $\dot{y} = -b\frac{x/a^2}{\sqrt{1-x^2/a^2}}$ в точках $x = \pm a$ равна $\dot{y} = \mp\infty$ и компьютерная программа выдаст сообщение об недопустимой операции (invalid result). Если же использовать параметрическое уравнение эллипса $x = a\cdot\cos(t);\ y = b\cdot\sin(t)$, то $\dot{x} = -a\cdot\sin(t);\ \dot{y} = b\cdot\cos(t)$ и никаких проблем в интервале $t \in [0, 2\pi]$ не возникает.

Периметр эллипса с $a = 5$ и $b = 1$ по формуле Рамануджана [8] составляет 21.01002…, а по процедуре LengthCurveP(...), реализующей (11) (см. Приложение 2), при n = 500 равен 21.01007… При этом следует заметить, что формула Рамануджана всегда дает отрицательную погрешность.

Для площади поверхности вращения от функции заданной в параметрическом виде следует использовать формулу

$$S = \int_{\alpha}^{\beta} y(t)\sqrt{\dot{x}^2(t) + \dot{y}^2(t)}\, dt, \qquad (12)$$

которая реализована в виде прцедуры SurfaceAreaP(...) в Приложении 2.

С целью проверки адекватности работы прикладных программ, приведенных в Приложении 2, была проведена верификация вычислительных алгоритмов на основе сравнения численных и аналитических решений, полученных для тестовых задач. В таблицах 2 и 3 приведены результаты для некоторые тест-функций. Расчеты проводились в операционной среде MacOS Monterey с CPU 2.3 GHz.

№	Функция	Интервал	n	Численное решение	Точное решение
1	$y = x^2$	$x \in [1, 2]$	5	3.167841…	3.167840…
2	$y = 2x^{3/2}$	$x \in [0, 11]$	20	73.935048…	74
3	$y = \ln(\cos(x))$	$x \in [0, \pi/3]$	50	1.316948…	1.316957…
4	$y = \mathrm{ch}(x)$	$x \in [1, 2]$	50	1.175200…	1.175201…
5	$y = \sqrt{1 - x^2} - \arcsin(x)$	$x \in [1, 8/9]$	100	1.885646…	1.885618…
6	$y = \frac{x^2}{4} - \frac{\ln(x)}{2}$	$x \in [1, e]$	50	2.097265…	2.097264…
7	$y = \sqrt{1 - x^2}$	$x \in [-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]$	500	1.570797…	1.570796…

Таблица 2. Результаты расчета длины дуги.

№	Функция	Интервал	n	Численное решение	Точное решение
1	$y = x$	$x \in [0, 1]$	1	4.442882…	4.442882…
2	$y = x^2$	$x \in [0, 1]$	10	3.809730...	3.809730…
3	$y = 2x^{3/2}$	$x \in [0, 11]$	100	16851.938…	16851.926 …

4	y =sin(x)	$x \in [0, \pi]$	500	14.423600..	14.423599 …
5	y =cos(x)	$x \in [0, \pi/2]$	100	7.211804…	7.211799…
6	$y = \sqrt{1 - x^2}$	$x \in$ [-0.7, 0.7]	500	8.796459…	8.796459…

Таблица 3. Результаты расчета площади поверхности вращения.

Общий анализ полученных вычислительных результатов и сопоставление их с другими численными методами показал, что описанный выше подход численного определения длины дуги и площади поверхности вращения даёт заметно лучшие результаты, чем методы Симпсона и Рунге-Кутты 4-го порядка.

Надо отметить, что представленные алгоритмы следует охарактеризовать как нересурсоемкие. Например, в следующей таблице для тестируемой функции y =sin(x) (см. табл. 3) при $x \in [0, \pi]$ представлены время расчета t в миллисекундах (мс) и относительная погрешность результата δ в зависимости от количества разчетных узлов (n+1):

n	100	200	300	400	500	600	700	800	900	1000
t (мс)	0.13…	0.16…	0.24…	0.29…	0.24…	0.29…	0.34…	0.39…	0.45…	0.49…
$\delta \cdot 10^6$	2.42…	0.63…	0.87…	0.18…	0.13…	0.22…	0.16…	0.07…	0.06…	0.05…

Полученные формулы и процедуры численного определения длины дуги и площади поверхности вращения, основанные на использовании типа данных Thdn и автоматического дифференцирования дают заметно лучшие результаты, чем известные методы численного интегрирования 4-го порядка погрешности, которые в некоторых случаях вызывают «пробуксовку». Как видно из кода компьютерных процедур (см. Приложение 1, 2 и 3) последнии относительно просто программируются.

Описанные выше процедуры численного определения длины дуги и площади поверхности вращения обладают универсальностью, позволяющей рассматривать функцию Y(X) как некий «черной ящик», поставляющий значение функции y(x) и точные величины ее первой и второй производных по данному значению $X = x + 1\boldsymbol{\varepsilon} + 0\boldsymbol{\omega}$.

Приложение 1.

Базовый вариант типа данных Thdn (truncated hyper-dual number). Язык програмирования Swift 5.2 (macOS 12.5).

```
public struct Thdn{
```

```
  var re, im1, im2: Double;
  init() {self.re = 0; self.im1 = 0; self.im2 = 0}
  init(re:Double) {self.re = re; self.im1 = 0; self.im2 = 0}
  init(re:Double,im1:Double,im2:Double) {self.re = re; self.im1 = im1; self.im2 = im2}
}

extension Thdn {
// OPERATORS (overloading):
  static prefix func - ( A:Thdn)->Thdn{return Thdn(re: -A.re, im1: -A.im1, im2: -A.im2)}
   static prefix func + ( A:Thdn)->Thdn{return A}
   static func + (A:Thdn, B:Thdn)->Thdn{return Thdn(re: A.re+B.re, im1: A.im1+B.im1,
         im2: A.im2+B.im2)}
   static func +=(lhs: inout Thdn, rhs: Thdn) {lhs = lhs + rhs}
   static func - (A:Thdn, B:Thdn)->Thdn{return Thdn(re: A.re-B.re, im1: A.im1-B.im1,
         im2: A.im2-B.im2)}
   static func -=(lhs: inout Thdn, rhs: Thdn) {lhs = lhs - rhs}
   static func *(A: Thdn, B: Thdn) -> Thdn {
     return  Thdn( re: A.re * B.re,
             im1: A.re * B.im1 + A.im1 * B.re,
             im2: A.re * B.im2 + 2.0*A.im1*B.im1 + A.im2 * B.re);
   }
   static func *(A: Thdn, B: Double) -> Thdn {
     return  Thdn( re:  A.re  * B,   im1:  A.im1 * B,   im2:  A.im2 * B);
   }
   static func *(A: Double, B: Thdn) -> Thdn {return B * A}
   static func *=(lhs: inout Thdn, rhs: Thdn) {lhs = lhs * rhs}
   static func *=(lhs: inout Thdn, rhs: Double) {lhs = lhs * rhs}
   static func /(A: Thdn, B: Thdn) -> Thdn {
     let b = B.re, b2 = b * b, b3 = b2 * b;
     return Thdn( re:  A.re / b,
             im1: (A.im1 / b - A.re * B.im1/b2),
             im2: (2.0*(A.re * B.im1 * B.im1 / b3 - A.im1 * B.im1 / b2) - A.re * B.im2 / b2 +
                   A.im2/b));
   }
   static func /(A: Thdn, B: Double) -> Thdn {
     return Thdn( re: A.re / B,   im1: (A.im1)/B,   im2: (A.im2)/B);
   }
   static func /=(lhs: inout Thdn, rhs: Thdn) {lhs = lhs / rhs}
// FUNCTIONS:
  static func inverse(A:Thdn)->Thdn{
    let a2 = A.re * A.re, a3 = a2 * A.re;
    return  Thdn( re: 1/A.re, im1: -A.im1/a2, im2: 2.0 * A.im1 * A.im1 / a3 - A.im2/a2);
  }
  static func pow(x:Double, n:Double)->Thdn{
    var Y = Thdn(re:  Darwin.pow(x, n),
             im1:  n * Darwin.pow(x, n - 1),
             im2:  n * (n - 1) * Darwin.pow(x, n - 2))
```

```
    if (Y. im1.isSignalingNaN){ Y.im1 = 0.0}
    if (Y. im2.isSignalingNaN){ Y.im2 = 0.0}
    return Y;
}
static func pow(X:Thdn, n:Double)->Thdn{
    return  self.HD(X: X, f: self.pow, n: n);
}
static func powX(a:Double, x:Double)->Thdn{
    let ln = Darwin.log(x), ax = Darwin.pow(a, x)
    return Thdn(re:  ax,    im1:  ax * ln,    im2:  ax * ln * ln);
}
static func log(x:Double)->Thdn{
    return Thdn( re:  Darwin.log(x),   im1: 1.0/x,  im2: -1.0/(x*x));
}
static func log(X:Thdn)->Thdn{return  self.HD(X: X, f: self.log)}
static func sqrt(x:Double)->Thdn{return self.pow(x: x, n: 0.5)}
static func sqrt(X:Thdn)->Thdn{return  self.HD(X: X, f: self.pow, n: 0.5)}
static func exp(x:Double)->Thdn{
    let ex = Darwin.exp(x);
    return Thdn( re: ex, im1: ex, im2: ex);
}
static func exp(X:Thdn)->Thdn{return  self.HD(X: X, f: self.exp)}
static func sin(x:Double)->Thdn{
    let s = Darwin.sin(x), c = Darwin.cos(x);
    return Thdn( re:  s,  im1:  c,   im2: -s);
}
static func sin(X:Thdn)->Thdn{return  self.HD(X: X, f: self.sin)}
static func cos(x:Double)->Thdn{
    let s = Darwin.sin(x), c = Darwin.cos(x)
    return Thdn( re:  c,    im1: -s,    im2: -c);
}
static func cos(X:Thdn)->Thdn{return  self.HD(X: X, f: self.cos)}
static func tan(x:Double)->Thdn{
    let t = Darwin.tan(x), t_1 = t*t + 1.0;
    return Thdn( re:  t,    im1: t_1,    im2: 2.0*t*t_1);
}
static func tan(X:Thdn)->Thdn{return  self.HD(X: X, f: self.tan)}
static func asin(x:Double)->Thdn{
    let t = Darwin.asin(x), t_1 = 1.0/Darwin.sqrt(1.0 - x*x);
    return Thdn( re:  t,   im1: t_1,     im2: x*Darwin.pow(t_1, -1.5));
}
static func asin(X:Thdn)->Thdn{return  self.HD(X: X, f: self.asin)}
static func acos(x:Double)->Thdn{
    let t = Darwin.acos(x), t_1 = -1.0/Darwin.sqrt( 1.0 - x*x);
    return Thdn( re:  t,    im1: t_1,     im2: -x*Darwin.pow(t_1, -1.5));
}
static func acos(X:Thdn)->Thdn{return  self.HD(X: X, f: self.acos)}
```

```
  static func atan(x:Double)->Thdn{
    let t = Darwin.atan(x), t0 = 1.0 + x*x,  t_1 = 1.0/t0;
    return Thdn( re:  t,   im1: t_1,    im2: -2.0*x/t0*t0);
  }
  static func atan(X:Thdn)->Thdn{return  self.HD(X: X, f: self.atan)}

  private static func HD(X:Thdn, f:(Double)->Thdn)->Thdn{return Thdn.ZX(F: f(X.re), X: X)}
  private static func HD(X:Thdn, f:(Double, Double)->Thdn, n:Double)->Thdn{
    return Thdn.ZX(F: f(X.re), X: X);
  }
  private static func ZX(F:Thdn,X:Thdn)->Thdn{
    return Thdn(re:   F.re,   im1:  X.im1 * F.im1,   im2:  X.im1*X.im1*F.im2 + X.im2*F.im1);
  }
}
```

Приложение 2.

Процедуры integral(...), LengthCurve(...), SurfaceArea(...), LengthCurveP(...) и SurfaceAreaP(...).

```
func integral(xo:Double, xn: Double, n:Int, F:(Thdn) -> Thdn) -> Double{
  let Δ = (xn - xo)/Double(n), h = Δ/2.0, p1 = 1.5, p2 = Δ/4.0, p3 = Δ*Δ/48.0;
  var f0 = 0.0;
  var J = 0.0,    F1:Thdn = F(Thdn(re:xo, im1:1, im2:0)),     F2:Thdn;
      J = F1.re*p1 + F1.im1*p2 + F1.im2*p3;
  for x in stride(from: xo + Δ, to: xn, by: Δ){
      f0 = F(Thdn(re:x - h, im1:1, im2:0)).re;
      F2 = F(Thdn(re:x, im1:1, im2:0));
      J +=  (F2.re*p1 + F2.im2*p3)*2.0 + f0;
      F1 = F2;
  }
  F2 = F(Thdn(re:xn, im1:1, im2:0));
  f0 = F(Thdn(re:xn - h, im1:1, im2:0)).re;
  J +=  (F2.re*p1 - F2.im1*p2 + F2.im2*p3 + f0);
  J *= (Δ/4.0);
  return J;
}

func LengthCurve(xo:Double, xn: Double, n:Int, Y:(Thdn) -> Thdn)-> Double{
  func Arc(X:Thdn)->Thdn{
    let T = Y(X);
    let fd = Thdn(re:T.im1, im1:T.im2, im2:0) ;
    return Thdn.pow(X: Thdn(re: 1.0) +  fd*fd, n: 0.5);
  }
  return integral(xo: xo, xn: xn, n: n, F: Arc);
}
```

```
func SurfaceArea(xo:Double, xn: Double, n:Int, Y:(Thdn) -> Thdn)-> Double{
  func Area(X:Thdn)->Thdn{
    let T = Y(X) ;
    let fd = Thdn(re:T.im1, im1:T.im2, im2:0) ;
    return T*Thdn.pow(X: Thdn(re: 1.0) +  fd*fd, n: 0.5);
  }
  return 2.0*Double.pi*integral(xo: xo, xn: xn, n: n, F: Area);
}

func LengthCurveP(xo:Double, xn: Double, n:Int, F:(Thdn) -> (X:Thdn, Y:Thdn))-> Double{
  func ArcP(X:Thdn)->Thdn{
    let T = F(X);
    let xd = Thdn(re:T.X.im1, im1:T.X.im2, im2:0);
    let yd = Thdn(re:T.Y.im1, im1:T.Y.im2, im2:0);
    return Thdn.pow(X: xd*xd +  yd*yd, n: 0.5);
  }
  return integral(xo: xo, xn: xn, n: n, F: ArcP);
}

func SurfaceAreaP(xo:Double, xn: Double, n:Int, F:(Thdn) -> (X:Thdn, Y:Thdn))-> Double{
  func ArcP(X:Thdn)->Thdn{
    let T = F(X);
    let xd = Thdn(re:T.X.im1, im1:T.X.im2, im2:0);
    let yd = Thdn(re:T.Y.im1, im1:T.Y.im2, im2:0);
    return T.Y*Thdn.pow(X: xd*xd +  yd*yd, n: 0.5);
  }
  return 2.0*Double.pi*integral(xo: xo, xn: xn, n: n, F: ArcP);
}
```

Приложение 3.

Примеры обращения к процедурам LengthCurve(...), SurfaceArea(...), LengthCurveP(...) и SurfaceAreaP(...).

```
// Design functions:
func LenX(X:Thdn)->Thdn{return Thdn.log(X: Thdn.cos(X: X))}
func AreaX(X:Thdn)->Thdn{return Thdn.sin(X: X)}
func Ellipse(T:Thdn)-> (X:Thdn, Y:Thdn){
  let  a = 5.0, b = 1.0;
  return (X:a*Thdn.cos(X: T), Y:b*Thdn.sin(X: T));
}
// Procedure call examples:
let Len = LengthCurve(xo: 0, xn: 11, n: 20, F: LenX);
let Area = SurfaceArea(xo: 0, xn: Double.pi, n: 200, F: AreaX);
```

```
let Lp = LengthCurveP(xo: 0, xn: 2.0*Double.pi, n: 1000, F: Ellipse);
let AreaEllipse =  SurfaceAreaP(xo: 0, xn: Double.pi, n: 1000, F: Ellipse);
```

Литература

1. *Korn G. A.*, *Korn T. M.* Mathematical Handbook for Scientists and Engineers: Definitions, Theorems, and Formulas for Reference and Review. - Mineola : Dover Publications, Inc., 2017. - 1130 p.
2. *Олифер В.И.* Усеченные гипер-дуальные числа в автоматическом дифференцировании.– 2020. URL:http://viosolutions.amerihomesrealty.com/pdf/Усеченные_гипер-дуальные_числа_в_автоматическом_дифференцировании.pdf
3. *Олифер В. И.* Численное интегрирование с использованием гипер-дуальных чисел.– 2021. URL: http://viosolutions.amerihomesrealty.com/pdf/Численное_интегрирование_с_использованием_гипер-дуальных_чисел.pdf
4. *Кеч В., Теодореску П.* Введение в теорию обобщённых функций с приложениями в технике. - М.: Мир, 1978, - 520 с.
5. *Naumann U.* The art of differentiating computer programs. Society for industrial and applied mathematics, Philadelphia, USA, 2012.
6. *Corliss G., Faure C., Griewank A., Hascoet L., Naumann U.* Automatic Differentiation Bibliography // Automatic Differentiation of Algorithms: From Simulation Optimization. Springer, 2002. p. 383- 425.
7. *Двайт Г. Б.* Таблицы интегралов и другие математические формулы. - М.: Наука, 1973. - 228 с.
8. *Villarino Mark B.* Ramanujan’s Perimeter of an Ellipse. Escuela de Matem´atica, Universidad de Costa Rica, 2008.

Абстракт

В данной публикации рассматривается применение метода численного интегрирования на основе разложения в ряд Тейлора и автоматического дифференцирования с использованием специальных дуальных чисел (усеченных гипер-дуальных чисел) для вычисления длины дуги и площади поверхности вращения.

Представлена компьютерная реализация этого метода для языка SWIFT операционной системы macOS. Проведены численные эксперименты.

Ключевые слова*: длина дуги, площадь поверхности вращения, эллиптические интегралы 2-го рода,численное интегрирование, усеченные гипер-дуальные числа, автоматическое дифференцирование, arc length, surface area of revolution, elliptic integrals of the 2nd kind, numerical integration, truncated hyper-dual numbers, automatic differentiation*

20 сентября 2022 г.