Информационное сообщение о программной
реализации АЛГЕБРЫ ДИФФЕРЕНЦИРОВАНИЯ

В Центральном экономико–математическом институте АН СССР разработана и реализована в языке BASIC на ЭВМ WANG программная система "TAYLOR". Эта система представляет собой компьютерную "Алгебру дифференцирования", позволяющую вычислять на ЭВМ **любое** требуемое число n производных от **произвольной** (данной) функции в заданной точке; функция может быть определена либо явной формулой $y = f(x)$, либо неявно – соотношением $\Phi(x, y) = 0$, либо параметрически – $x = f_1(t)$, $y = f_2(t)$, либо, наконец, с помощью дифференциального уравнения $y' = G(x, y)$, $y(x_0) = y_0$ (задача Коши). Во всех случаях абоненту достаточно лишь написать в языке BASIC **формулу** соответствующей функции ($f(x)$; $\Phi(x, y)$; $f_1(t)$, $f_2(t)$; $G(x, y)$) и указать значения аргументов и порядок дифференцирования n.

Речь идет о вычислении производных не путем применения приближенных разностных схем, а непосредственно на основе формул дифференциального исчисления, точность результата ограничивается лишь возможностями ЭВМ. Это означает, что **система Taylor открывает принципиально новые точные и простые средства** для широкого использования в вычислительной практике методов, связанных с представлением функций рядами Тейлора, методом асимптотических разложений, прямых методов высших производных. В этом отношении система TAYLOR является уникальной и не имеет аналогов в мировой практике.

Система находится в состоянии демонстрационной готовности.

АЛГЕБРА ДИФФЕРЕНЦИРОВАНИЯ
проспект программной системы

Комплекс программ Алгебры дифференцирования TAYLOR включает процедуру двух уровней. Процедуры первого, базового уровня являются фундаментом всей системы и имеют важное самостоятельное значение – они открывают принципиально новые возможности в вычислительной математике. Процедуры второго уровня реализуют часть этих возможностей в форме стандартных программ для решения типовых задач. Все процедуры реализованы в языке BASIC на ЭВМ WANG.

1. Базовые процедуры

Базовые процедуры осуществляют вычисление производных данной функции одного или двух аргументов в заданной точке и в требуемом количестве. В обращении к процедурам параметром, задающим функцию, является строчная символьная переменная, содержащая запись формулы данной функции в обычных символах языка BASIC (аналогично правой части оператора присваивания). Поскольку вычисление производных в заданной точке равносильно построению отрезка ряда Тейлора в окрестности этой точки, мы даем описание процедур непосредственно в виде процедур вычисления коэффициентов ряда Тейлора – это более естественно с точки зрения приложений.

A. Функции одного переменного

1.1. Proc 11 ("$f(x)$"; x_0, n) – разложение функции в ряд Тейлора (вычисление производных). Процедура вычисляет коэффициенты $c_k = c_k(f; x_0)$ отрезка ряда Тейлора функции $f(x)$ в точке $x = x_0$

$$f(x) \sim c_0 + c_1(x - x_0) + \ldots + c_n(x - x_0)^n, \quad c_k := \frac{1}{k!}\,\frac{\mathrm{d}^k f}{\mathrm{d}x^k}|_{x_0}\ ;$$

здесь f – произвольная дифференцируемая функция, заданная формулой "$f(x)$", записанной в языке BASIC, n – требуемый порядок разложения, x_0 – данное значение аргумента функции f.

1.2. Proc 12 ("$f(x)$"; x_0, n) – разложение обратной функции. Процедура вычисляет коэффициенты Тейлора $c_k(g; u_0)$ функции $x = g(u)$, обратной к функции $u = f(x)$ в точке $u_0 := f(x_0)$ (такая функция в окрестности точки u_0 существует и единственна, если $f'(x_0) \neq 0$).

1.3. Proc 13 ("$f_1(t)$", "$f_2(t)$"; t_0, n) – разложение функции, заданной параметрически. Процедура вычисляет коэффициенты Тейлора $c_k(y; x_0)$ функции $y(x)$, определенной параметрическими соотношениями $x = f_1(t)$, $y = f_2(t)$, в точке $x_0 := f_1(t_0)$ (такая функция в окрестности точки x_0 существует и единственна, если $f_1'(t_0) \neq 0$).

Б. Функции двух переменных

1.4. Proc 14 ("$\Phi(x,y)''$; x_0, y_0, n) – разложение (вычисление частных производных) функции двух переменных. Процедура вычисляет коэффициенты $c_{kt}(\Phi;\ x_0, y_0)$ отрезка порядка n "двумерного" ряда Тейлора функции $\Phi(x,y)$ в точке (x_0, y_0)

$$\Phi(x,y) \sim \sum_{0\le k+t\le n} c_{kt}(x-x_0)^k(y-y_0)^t, \quad c_{kt} := \frac{1}{k!t!}\frac{\partial^{k+t}\Phi}{\partial x^k \partial u^t}|_{(x_0,y_0)} \quad .$$

Смысл параметров процедуры аналогичен описанию Proc 11.

1.5. Proc 15 ("$\Phi(x,y)''$, "$y(x)''$; x_0, n) – разложение "полной" функции (вычисление полных прооизводных). Процедура вычисляет коэффициенты Тейлора $c_k(f; x_0)$ "полной" функции одной переменной $f(x) := \Phi(x, y(x))$ в точке x_0.

1.6. Proc 16 ("$\Phi(x,y)''$; x_0, y_0, n) – разложение (вычисление производных) неявной функции. Процедуры вычисляют коэффициенты Тейлора $c_k(y; x_0)$ неявной функции $y(x)$, определенной соотношением $\Phi(x,y) = \Phi(x_0, y_0)$ в точке x_0 (такая функция в окрестности точки x_0 существует и единственна, если $\dfrac{\partial\Phi}{\partial y}|_{(x_0,y_0)} \neq 0$).

2. Процедуры решения типовых задач

Комплект процедур этого уровня может пополняться неограниченно. Здесь мы приводим несколько типовых задач, решение которых дается непосредственным применением базовых процедур.

2.1. Proc 21 ("$f(x)''$; x_0, n) – вычисление коэффициентов Тейлора $c_k(F; x_0)$ первообразной $F(x) := \int\limits_{x_0}^{x} f(t)dt$. Эта процедура может использоваться, в частности, для вычисления интеграла (определенного)

$$\int\limits_{x_0}^{a} f(t)dt = F(a) \simeq \sum_{k=1}^{n} c_k(F; x_0)(a-x_0)^k \quad .$$

2.2. Proc 22 ("$f(x)''$; x_0, n) – нахождение корня x^* уравнения $f(x) = 0$, лежащего в окрестности данной точки x_0. Решение уравнения $f(x) = 0$ интерпретируется как задача вычисления функции $x = g(u)$, обратной к функции $u = f(x)$ в точке $u = 0$, т. е. $x^* = g(0)$; эта задача решается с использованием процедуры разложения 1.2 в точке $u_0 := f(x_0)$.

2.3. Proc 23 ("$\Phi(x,y)''$; x_0, y_0, n) – решение задачи Коши для дифференциального уравнения $y' = \Phi(x,y)$, $y(x_0) = y_0$. В окрестности точки x_0 искомая функция представляется отрезком ряда Тейлора длины n; коэффициенты разложения вычисляются данной процедурой.

2.4. Proc 24 ("$f(x)''$; x_0, λ, n) – вычисление интеграла от быстроосциллирующей функции

$$I(\lambda) := \lambda \int_{x_0}^{\infty} f(x) \sin(\lambda x)\, \mathrm{d}x\ , \quad \lambda \gg 1$$

(предполагается, что $f(x)$ убывает на бесконечности экспоненциально). Используется асимптотическое разложение

$$I(\lambda)\ \sim\ \sum_{k=0}^{n} \frac{b_k}{\lambda^k} \quad \lambda \to \infty \quad ,$$

где b_k явно выражается через коэффициенты Тейлора $c_k(f; x_0)$ функции f. Интегралы типа $I(\lambda)$ часто встречаются в задачах математической физики (волновая механика, оптика, акустика и др.).

Преамбула

В настоящем Сообщении приводятся алгоритмы, решающие следующие (основные) задачи.

A1. Вычисление производных сложной функции. По известным значениям производных $u', u'', \ldots, u^{(n)}$ данной функции $u(x)$ в некоторой точке x_0 и производных $f', f'', \ldots, f^{(n)}$ данной функции $f(u)$ в точке $u_0 = u(x_0)$ находятся значения производных $\Psi', \Psi'', \ldots, \Psi^{(n)}$ сложной функции $\Psi(x) = f(u(x))$ в точке x_0. Специально выделены случаи $f(u) = 1/u$, $f(u) = \mathrm{e}^u$, допускающие более простое решение.

A2. Вычисление производных обратной функции. По известным значениям производных $f', f'', \ldots, f^{(n)}$ данной функции $f(u)$ в некоторой точке u_0 находятся значения $u', u'', \ldots, u^{(n)}$ производных обратной функции $u(x)$ в точке $x_0 = f(u_0)$. Под обратной функцией понимается такая функция $u(x)$, определенная и непрерывная в некоторой окрестности точки x_0, что $u(x_0) = u_0$, $f(u(x)) \equiv x$.

A3. Вычисление частных производных сложной функции двух переменных. По известным значениям частных производных $\dfrac{\partial^{p+q}\Phi}{\partial x^p \partial u^q}$, $1 \le p+q \le n$ данной функции $\Phi(x,u)$ в некоторой точке (x_0, u_0) и производных $f', f'', \ldots, f^{(n)}$ данной функции $f(z)$ в точке $z_0 = \Phi(x_0, u_0)$ находятся значения частных производных $\dfrac{\partial^{p+q}\Psi}{\partial x^p \partial u^q}$, $1 \le p+q \le n$ сложной функции $\Psi(x,u) = f(\Phi(x,u))$ в точке (x_0, u_0).

A4. Вычисление полных производных. По известным значениям производных $u', u'', \ldots, u^{(n)}$ данной функции $u(x)$ в некоторой точке x_0 и значениям частных производных $\dfrac{\partial^{p+q}\Phi}{\partial x^p \partial u^q}$, $1 \le p+q \le n$ данной функции $\Phi(x,u)$ в точке (x_0, u_0), $u_0 = u(x_0)$ находятся производные $\Psi', \Psi'', \ldots, \Psi^{(n)}$ функции $\Psi(x) = \Phi(x, u(x))$ в точке x_0.

A5. Вычисление производных неявной функции. По известным значениям частных производных $\dfrac{\partial^{p+q}\Phi}{\partial x^p \partial u^q}$, $1 \le p+q \le n$ данной функции $\Phi(x,u)$ в некоторой точке (x_0, u_0), удовлетворяющей условию $\Phi(x_0, u_0) = 0$, находятся производные $u', u'', \ldots, u^{(n)}$ неявной функции $u(x)$ в точке x_0; под неявной понимается такая функция $u(x)$, определенная и непрерывная в некоторой окрестности точки x_0, что $u(x_0) = u_0$, $\Phi(x, u(x) \equiv 0$.

Всюду здесь n – **любое** (но фиксированное) натуральное число, данные функции u, f, Φ предполагаются дифференцируемыми требуемое число раз, в остальном они могут быть произвольными. Следует указать, что все перечисленные алгоритмы находят точные значения производных по формулам дифференциального исчисления (в отличие от конечно-разностных методов, дающих лишь приближенные значения и работающих лишь при малых n). При этом, как аргумент x, так и функции u, f, Φ могут быть комплекснозначными. Алгоритмы приведены в тексте в записи, представляющей собой сокращение языка Алгол, и, будучи реализованы в виде процедур Алгола, прошли проверку на ЭВМ; в этих процедурах реализован случай вещественнозначных функций, точные тексты процедур даны в Приложении. Отметим, что все алгоритмы А1–А6 рекуррентны, поэтому при вычислении производных некоторого порядка n вычисляются и все производные меньших порядков, независимо от того, нужны они абоненту или нет.

То обстоятельство, что для элементарных функций

$$x^p\ (p = const),\ \mathrm{e}^x,\ \ln x,\ \sin x,\ \cos x$$

производные любого порядка выписываются в явном виде через эти же функции, а также наличие названных элементарных функций в составе математического обеспечения ЭВМ позволяют непосредственно получить на ЭВМ значения производных любых порядков от этих функций. Применяя затем многократно алгоритм **А1** и формулы дифференцирования

$$(\lambda u)^{(k)} = \lambda u^{(k)}, \quad (u+v)^{(k)} = u^{(k)} + v^{(k)} \tag{1}$$

$$(uv)^{(k)} = \sum_{j=0}^{k} \mathsf{C}_k^j u^{(j)} v^{(k-j)} \tag{2}$$

можно получить на ЭВМ производные любого порядка от функций, задаваемых любым арифметическим выражением, составленным из элементарных функций (арифметических функций). Алгоритм **А2** позволяет вычислять производные от функций, обратных к арифметическим, а алгоритм **А5** (вместе с алгоритмом **А3**) – от функций, задаваемых арифметическим уравнением $\Phi(x, u) = 0$. Таким образом, для ЭВМ оказывается доступной **алгебра дифференцирования** (отметим, что в эту алгебру можно включать и специальные функции, если только указать способ вычисления их производных).

Простота предлагаемых алгоритмов и универсальность задачи вычисления производных высших порядков (особенно в инженерно–технических расчетах) ставят в повестку дня вопрос о внедрении алгебры дифференцирования в практику **непосредственно через языки программирования**. Речь идет, таким образом, о соответствующем **расширении универсальных языков программирования (Алгол, Фор-**

тран, PL и др.) и трансляторов с них. Дело должно быть доведено до такой стадии, чтобы для получения на ЭВМ, например, производных функции

$$\ln\left(\sqrt{1+x^2}-\sin\ x^{2/3}\right), \quad x_0=8$$

или производных неявной функции $u(x)$, определяемой уравнением

$$u^2-2\sin\,(xu)+\ln\,(x+u)=0, \quad (x_0,u_0)=(1,0)$$

достаточно было бы написать в программе соответствующие выражения, указать значения аргументов, число n требуемых производных и адрес засылки результата (подобно тому, как это имеет место сейчас в арифметических выражениях в операторах присваивания). Можно, конечно, реализовать предлагаемые алгоритмы в виде процедур (например, как это сделанр в Приложении), но тогда решение названных задач для абонента значительно усложняется, так как потребуется многократное обращение к этим процедурам, что существенно удлиняет программу и, кроме того, не исключает вероятности внесения ошибки.

Представляется, что внедрение алгебры дифференцирования может открыть новые потенциальные возможности в разработке методов вычислительной математики. Например, задачу нахождения корня уравнения $f(u)=0$ (здесь u – независимая переменная) можно интерпретировать как задачу вычисления значения обратной функции $u(x)$ в точке $x=0$. Беря некоторое начальное приближение u_0 и полагая $x_0:=f(u_0)$, можно с помощью алгоритма **A2** найти производные $u',u'',\ldots,u^{(n)}\,|_{x_0}$ в количестве, достаточном для того, чтобы остаточным членом ряда Тейлора в точке x_0

$$\begin{aligned} u^* \;=\; u(x)|_{x=0} \;&=\; u_0+(x-x_0)u'+\tfrac{1}{2}(x-x_0)^2u''+\ldots|_{x=0}= \\ &=\; u_0-x_0u'+\tfrac{1}{2}x_0^2u''-\ldots \end{aligned} \qquad (3)$$

можно было бы пренебречь (в предположении, что радиус сходимости ряда (3) больше $|x_0|$, для этого начальное приближение должно быть достаточно хорошим, в частности, функция $f(u)$ должна быть монотонна в интервале (u_0,u^*); отметим. что известный метод Ньютона ограничивается двумя первыми членами ряда (3) с последующим итерированием). Мы получаем фактически прямой метод решения уравнения $f(u)=0$. К этой же задаче сводится вычисление значения неявной функции $u(x)$, определяемой уравнением $\Phi(x,u)=0$, в данной точке $x=\xi$; для этого надо рассмотреть уравнение $f(u)=0$, где $f(u):=\Phi(\xi,u)$.

Другой пример: для интегралов вида

$$\varphi(x)=\int_0^\infty e^{-xt}f(t)\mathrm{d}t$$

(интегралы Лапласа, широко применяемые в математической физике) при $x \to \infty$ имеет место асимптотическое разложение

$$\varphi(x) \sim \sum_k f^{(k)}(0) x^{-(k+1)} \tag{4}$$

(см. [1, с.41]). Вообще, во всех методах, использующих разложение в ряды, знание производных высших порядков оказывается весьма полезным.

Важной областью применения алгебры дифференцирования могут стать дифференциальные уравнения. Например, в задаче Коши (относительно неизвестной функции $u(x)$)

$$u' = \Phi(x, u), \quad u(x_0) = u_0$$

знание частных производных $\dfrac{\partial^{p+q}\Phi}{\partial x^p \partial u^q}|_{(x_0,u_0)}$ до порядка n (они могут быть найдены с помощью алгоритмов **A1**, **A3**) и вытекающее из данного уравнения рекуррентное соотношение

$$u^{(k+1)}(x) = \Psi^{(k)}(x), \qquad \Psi(x) := \Phi(x, u(x)) \quad ,$$

позволяют, многократно применяя алгоритм **A4**, получить последовательно значения $u', u'', \ldots, u^{(n)}\ |_{x=x_0}$ и тем самым представление искомой функции $u(x)$ в виде отрезка ряда Тейлора длины n; это реализовано в Приложении – см. процедуру dif 6.

Отметим, что идея использования дифференциальных соотношений для вычисления высших производных и тем самым для разложения решения задачи Коши в ряд Тейлора уже известна в литературе. Впервые она была сформулирована Муром в 1966 году [2], и затем получила развитие в работах [3–4]; в недавней работе [5] соответствующий метод назван методом быстрого разложения Тейлора (б.р.Т.) и там же отмечается, что его применение к задаче Коши может сократить время счета в 5–50 раз. Эти данные характеризуют эффективность использования рядов Тейлора и в этом смысле указанную оценку можно отнести и к процедуре dif 6. Однако применение метода б.р.Т. доступно в каждом конкретном случае только при неформальном участии абонента, а создание на его основе универсального пакета программ для решения задачи Коши сложно. Дело в том, что этот метод требует приведения задачи к так называемой канонической форме – к системе дифференциальных уравнений первого порядка с примитивными правыми частями, содержащими лишь одну арифметическую операцию $+, -, \cdot, /$. Такую редукцию в общем случае трудно формализовать, и попытки создать общую компилирующую программу до сих пор, повидимому, не увенчались успехом. Утверждение в [5, с. 1095], что это уже сделано в [3,4] не подтверждается: проверка показала, что там имеются лишь иллюстративные примеры и

изложены принципы построения общего компилятора; однако с момента публикации [4] прошло уже более 10 лет, а сообщений о создании такого компилятора не появилось (кустарную работу [6] мы не принимаем во внимание). Компилятор, требуемый при нашем подходе, практически ничем не отличается от уже действующего в трансляторах компилятора арифметического выражения, и он может быть создан в короткие сроки (конечно, при участии специалистов).

С появлением данных базовых алгоритмов задача создания компилятора для реализации на ЭВМ алгебры дифференцирования становится реальной и актуальной.

Наконец, упомянем о важной задаче вычисления высших производных функции $y = y(x)$, заданной параметрически: $y = f(t),\ \ x = h(t)$; требуется, зная $f', f'', \ldots, f^{(n)} \ |_{t_0},\ \ h', h'', \ldots, h^{(n)} \ |_{t_0}$, найти $y', y'', \ldots, y^{(n)} \ |_{x_0},\ x_0 = h(t_0)$, такая задача рассматривается в [7, П.3], где выводится красивая формула, выражающая $y^{(n)}$ в форме определителя n—го порядка. В рамках настоящего Сообщения эта задача решается применением двух базовых алгоритмов **A1** и **A2**: поскольку $y(x)$ можно рассматривать как сложную функцию $y = f(t),\ t = t(x)$, где $t(x)$ – функция, обратная к $h(t)$, то сначала с помощью алгоритма **A2** находятся значения $t', t'', \ldots, t^{(n)} \ |_{x_0}$ (для этого используются только $h', h'', \ldots, h^{(n)} \ |_{t_0}$), затем по ним и значениям $f', f'', \ldots, f^{(n)} \ |_{t_0}$ алгоритм **A1** находит искомые $y', y'', \ldots, y^{(n)} \ |_{x_0}$.

* * *

В основном тексте Сообщения приняты следующие обозначения и соглашения:

1) $k! = 1 \cdot 2 \cdots k$ при целом $k \geq 1, \quad 0! = 1$;

2) $y^{(k)}$ – k—я производная функции $y(x)$
$k = 1, 2, \ldots, \quad y^{(0)} = y$;

3) если $m > n$ то сумма $\sum\limits_{j=m}^{n} a_j$ считается равной нулю.

Для записи алгоритмов используется сокращенная форма языка Алгол. Оператор цикла

for j=m **step** 1 **until** n **do begin**... **end**

записывается так:

$$\overset{n}{\underset{j=m}{\Big\{ \ \ldots \ \Big\}}} \ ,$$

при этом, если $m > n$, то этот оператор эквивалентен пустому. Оператор цикла

for j=m **step** -1 **until** n **do begin**... **end**

записывается в виде

$$\left\{ \begin{array}{c} \\ \dots \\ \\ \end{array} \right\}_{n}^{j=m} \quad ,$$

при этом, если $m < n$, то этот оператор эквивалентен пустому.

1. Факторизованные производные

Обратимся к формуле Лейбница (2); она упрощается, если вместо обычных производных пользоваться *факторизованными*. Таковыми мы называем производные, обозначаемые нижним индексом (k), которые по определению полагаются равными

$$y_{(k)} := \frac{1}{k!} y^{(k)} \ , \quad k = 0, 1, 2, \dots \quad .$$

Для факторизованных производных формулы (1) сохраняются, а формула (2) примет вид свертки

$$(uv)_{(k)} = \sum_{j=0}^{k} u_{(j)} v_{(k-j)} \quad , \tag{5}$$

поскольку $\mathsf{C}_k^j = \dfrac{k!}{j!(k-j)!}$. Отметим, кроме того, что $u_{(0)} = u$, а также формулу

$$\left(u_{(1)}\right)_{(j)} = \frac{1}{j!} \left(u_{(1)}\right)^{(j)} = \frac{1}{j!} \left(u^{(1)}\right)^{(j)} = \frac{u^{(j+1)}}{j!} = (j+1) u_{(j+1)} \quad . \tag{6}$$

Приведем примеры использования этих формул для построения простейших рекуррентных алгоритмов вычисления производных высших порядков.

Пример 1.

Применим (5) для вычисления производных $v_{(0)}, v_{(1)}, \dots, v_{(n)} \,|_{x_0}$ функции $v(x) = 1/u(x)$, исходя из значений $u_{(0)}, u_{(1)}, \dots, u_{(n)} \,|_{x_0}$; т. к. $uv \equiv 1$, то $(uv)_{(k)} = 0$ при $k \geq 1$, и согласно (5)

$$0 = u_{(0)} v_{(k)} + u_{(1)} v_{(k-1)} + \dots + u_{(k)} v_{(0)} \quad ,$$

откуда

$$\left\{ \begin{array}{rcl} v_{(k)} & = & -\dfrac{1}{u_{(0)}} \displaystyle\sum_{j=1}^{k} u_{(j)} v_{(k-j)} \ , \quad k = 1, 2, \dots \\ \\ v_{(0)} & = & \dfrac{1}{u_{(0)}} \quad . \end{array} \right.$$

Это рекуррентное соотношение позволяет последовательно друг за другом получить значения $v_{(0)}, v_{(1)}, \ldots, v_{(n)}$ при известных $u_{(0)}, u_{(1)}, \ldots, u_{(n)}$, что и требуется. Соответствующую последовательность вычислений можно записать в виде следующего алгоритма

вход $(n;\ a\,[0:n])$; рабочий массив $b\,[0:n]$;
if $a_0 = 0$ **then goto** M; $b_0 := 1/a_0$

$$\overset{n}{\underset{k=1}{\left\{ b_k := -b_0 * \sum_{j=1}^{k} a_j * b_{k-j} \right\}}} \qquad \overset{n}{\underset{k=0}{\left\{ a_k := b_k \right\}}}$$

M: выход (a)

Пояснение. На входе $a_k = u_{(k)},\ k = 0, 1, \ldots, n$; на выходе, если $a_0 = 0$, то функция v не существует (в этом случае массив a тот же, что на входе), если $a_0 \neq 0$, то $a_k = v_{(k)},\ \ k = 0, 1, \ldots, n$. Укажем также, что наряду с $n = 1, 2, \ldots$ допустимо и значение $n = 0$. При (относительно) больших n алгоритм требует $\sim n^2/2$ операций, считая, что операция = умножение + сложение; при $n = 10$ это ~ 50 операций.

Аналогичный прием годится и для частного двух функций $v(x) = g(x)/u(x)$; применяя к равенству $g = uv$ формулу (5), получим

$$g_{(k)} = u_{(0)}v_{(k)} + u_{(1)}v_{(k-1)} + \ldots + u_{(k)}v_{(0)} \quad ,$$

откуда

$$v_{(k)} = \frac{1}{u_{(0)}} \left(g_{(k)} - \sum_{j=1}^{k} u_{(j)}v_{(k-j)} \right), \quad k = 0, 1, \ldots \quad .$$

Это рекуррентное соотношение позволяет последовательно получить значения $v_{(0)}, v_{(1)}, \ldots, v_{(n)} \ |_{x_0}$ при известных $u_{(0)}, u_{(1)}, \ldots, u_{(n)} \ |_{x_0}$ и $g_{(0)}, g_{(1)}, \ldots, g_{(n)} \ |_{x_0}$. Запишем соответствующий алгоритм

вход $(n;\ a, b\,[0:n])$; **if** $a_0 = 0$ **then goto** M;

$$\overset{n}{\underset{k=0}{\left\{ b_k := \frac{1}{a_0} \left(b_k - \sum_{j=1}^{k} a_j * b_{k-j} \right) \right\}}}$$

M: выход (a_0, b)

Пояснение. На входе $a_k = u_{(k)},\ \ b_k = g_{(k)},\ \ k = 0, 1, \ldots, n$; на выходе, если $a_0 = 0$, то функция v не существует, если $a_0 \neq 0$, то $b_k = v_{(k)}$. Допустимо $n \geq 0$, оценка числа операций та же, что в предыдущем случае.

Пример 2.

Решаем ту же задачу, что и в примере 1 для функции $v(x) = \exp(u(x))$; т. к. $v_{(1)} = v \cdot u_{(1)}$, то согласно (5), (6) при $k \geq 1$ имеем

$$\begin{aligned} v_{(k)} &= \frac{1}{k}\left(v_{(1)}\right)_{(k-1)} = \frac{1}{k}\left(vu_{(1)}\right)_{(k-1)} = \\ &= \frac{1}{k}\sum_{j=0}^{k-1}\left(u_{(1)}\right)_{(j)} v_{(k-1-j)} = \frac{1}{k}\sum_{j=0}^{k-1}(j+1)u_{(j+1)}v_{(k-1-j)} = \\ &= \frac{1}{k}\sum_{j=1}^{k} j u_{(j)} v_{(k-j)} \quad . \end{aligned} \tag{7}$$

Это соотношение также рекуррентно и (с учетом того, что $v_{(0)} = \exp\left(u_{(0)}\right)$) позволяет вычислить $v_{(0)}, v_{(1)}, \ldots, v_{(n)}$ при известных $u_{(0)}, u_{(1)}, \ldots, u_{(n)}$. Алгоритм запишется в виде

вход $(n;\ a\ [0:n])$; рабочий массив $b\ [1:n]$;

$$a_0 := \exp(a_0) \qquad \overset{n}{\underset{k=1}{\Bigg\{}} b_k := k * a_k \qquad a_k := \frac{1}{k}\sum_{j=1}^{k} b_j * a_{k-j} \Bigg\}$$

выход (a) .

Пояснение. На входе $a_k = u_{(k)}$, на выходе $a_k = v_{(k)},\ k = 0, 1, \ldots, n;\ n \geq 0$, число операций $\sim n^2/2$.

2. Вычисление производных сложной функции

Примеры 1 и 2 являются специальными (допускающими простое решение) случаями общей задачи вычисления производных сложной функции; они приведены нами как иллюстрация принципа рекуррентности. В данном пункте мы обратимся к общему случаю, именно здесь заложена основная идея нашего метода, которая проводится затем и во всем последующем.

Пусть задана сложная функция $f(u),\ u = u(x)$; зафиксировав функцию $u(x)$, будем обозначать через $L_k(g)$ полную $k-$ю факторизованную произвводную произвольной сложной функции $g(u)$, т. е.

$$L_k(g) := \frac{1}{k!}\frac{\mathrm{d}^k g}{\mathrm{d}x^k}, \quad g = g(u),\ u = u(x),\ k = 0, 1, \dots \quad . \tag{8}$$

Наша задача сейчас – построить алгоритм вычисления $(L_0, L_1, \dots, L_n)_f|_{x=x_0}$ при известных $u_{(0)}, u_{(1)}, \dots, u_{(n)}|_{x_0}$ и $f^{(0)}, f', f'', \dots, f^{(n)}|_{u_0=u(x_0)}$; обратим внимание, что от f мы используем обычные производные.

Применим к равенству

$$L_1(g) = g'u' = g'u_{(1)}$$

формулы (5), (6); получим при $k \geq 1$

$$\begin{aligned} L_k(g) &= \frac{1}{k}L_{k-1}\big(L_1(g)\big) = \frac{1}{k}L_{k-1}\big(g'u_{(1)}\big) = \\ &= \frac{1}{k}\sum_{j=0}^{k-1}\big(u_{(1)}\big)_{(j)}L_{k-1-j}(g') = \\ &= \frac{1}{k}\sum_{j=0}^{k-1}(j+1)u_{(j+1)}L_{k-1-j}(g') = \\ &= \frac{1}{k}\sum_{j=1}^{k} j u_{(j)}L_{k-j}(g') \quad . \end{aligned} \tag{9}$$

Подставляя сюда в качестве g функцию $f^{(m)}$, получим

$$\begin{cases} L_k\big(f^{(m)}\big) = \dfrac{1}{k}\displaystyle\sum_{j=1}^{k} j u_{(j)}L_{k-j}\big(f^{(m+1)}\big) \quad k = 1, 2, \dots \\ L_0\big(f^{(m)} = f^{(m)} \quad . \end{cases} \tag{10}$$

Отсюда следует, что можно вычислить $(L_0, L_1, \ldots, L_n)\ f$, если известны $(L_0, L_1, \ldots, L_{n-1})\ f'$ (полагаем в (10) $m = 0$,) последние, в свою очередь, можно найти, если известны $(L_0, L_1, \ldots, L_{n-2})\ f''$ (полагаем $m = 1$), и так далее, в общем случае необходимо знать $(L_0, L_1, \ldots, L_{n-m})\ f^{(m)}$ и, в конце концов, при $m = n$ только $L_0\big(f^{(n)}\big) = f^{(n)}$. Таким образом, решение задачи сводится к построению "пирамиды" значений

$$\begin{array}{l} L_0\big(f^{(n)}\big) \\ L_1\big(f^{(n-1)}\big),\ L_0\big(f^{(n-1)}\big) \\ \ldots\ldots\ldots\ldots \\ L_n\big(f\big),\ L_{n-1}\big(f\big),\ \ldots,\ L_0\big(f\big) \end{array} \tag{11}$$

(в данной точке $x = x_0$); отметим, что пирамида строится "ходом назад" от больших значений m к меньшим. Описанная схема вычислений представлена следующим рекуррентным алгоритмом.

Алгоритм A1.

вход $(n;\ a, b\ [0:n])$ $\displaystyle\overset{n}{\underset{i=1}{\Big\{}} a_i := i * a_i \quad m := n - i$

$\displaystyle\overset{i}{\underset{k=1}{\Big\{}} p := k + m \quad b_p := \frac{1}{k}\sum_{j=1}^{k} a_j * b_{p+1-j} \Big\}\Big\}$

$\displaystyle\overset{n}{\underset{j=0}{\Big\{}} a_j := b_j \Big\}$ выход(a)

Пояснения.

1) На входе $a_j = u_{(j)},\ b_j = f^{(j)}$, на выходе

$$a_j = L_j(f),\ j = 0, 1, \ldots, n\ ; \quad n \geq 0 \quad .$$

2) В целях экономии памяти алгоритм построен таким образом, что значения $L_k\big(f^{(m)}\big)$ записываются в ту же ячейку, что и $L_{k-1}\big(f^{(m+1)}\big)$, т. е. в ячейку b_{k+m}.

3) Заключительный перенос результатов в массив a сделан для того, чтобы удобно было пользоваться алгоритмом **A1** для вычисления производных суперпозиции нескольких функций; например, если нужно найти производлные функции $f_3\big(f_2\big(f_1(u)\big)\big)$, то при первом обращении к **A1** нужно подготовить массив a и b, а при втором и третьем только массив b, т. к. a будет уже подготовлен.

4) Значения $a_0,\ b_0$ для алгоритма безразличны, они введены для удобства абонента; алгоритм требует $\sim n^3/6$ операций, при $n = 10$ это ~ 170 операций.

3. Вычисление производных обратной функции

Пусть задана функция $f(u)$ (u – независимый аргумент), достаточное число раз дифференцируемая в данной точке u_0; положим $x_0 := f(u_0)$. Под обратной функцией понимается функция $u(x)$, определенная в некоторой окрестности точки x_0 и столько же раз дифференцируемая в самой точке x_0, такая, что $u(x_0) = u_0$ и $f(u(x)) \equiv x$. Например, если $u_0 = -2$, $f(u) = u^2 - 1$, то $x_0 = 3$, $u(x) = -\sqrt{1+x}$. Обратная функция существует (и при этом единственна) тогда и только тогда, когда $f'(u_0) \neq 0$; предполагаем, что это условие выполнено. Наша задача сейчас – построить алгоритм вычисления $f(u_0) = u', u'', \ldots, u^{(n)} \mid_{x_0 = f(u_0)}$ при известных $f', f'', \ldots, f^{(n)} \mid_{u_0}$.

Пусть $u(x)$ – обратная функция, и операторы L_k определены в (8), тогда $L_1(f) = 1$, $L_2(f) = L_3(f) = \ldots = L_n(f) = 0$. Из выражений

$$
\begin{aligned}
L_1(f) &= f^{(1)} u_{(1)} \\
L_2(f) &= \frac{1}{2} f^{(2)} u_{(1)}^2 + f^{(1)} u_{(2)} \\
L_3(f) &= \frac{1}{6} f^{(3)} u_{(1)}^3 + f^{(2)} u_{(1)} u_{(2)} + f^{(1)} u_{(3)} \\
&\ldots\ldots\ldots\ldots
\end{aligned}
$$

легко усмотреть, что $t-$я производная $u_{(t)}$ появляется впервые в выражении для $L_t(f)$ и входит в него с множителем $f^{(1)} = f'(u_0) \neq 0$. Отсюда следует, что $u_{(1)}$ можно найти из равенства $L_1(f) = 1$, после этого, зная $u_{(1)}$, можно найти $u_{(2)}$ из равенства $L_2(f) = 0$, затем, зная $u_{(1)}$, $u_{(2)}$, можно найти $u_{(3)}$ из равенства $L_3(f) = 0$ и так далее; вообще, зная $u_{(1)}, u_{(2)}, \ldots, u_{(t-1)}$, можно найти $u(t)$ из равенства $L_t(f) = 0$. Важно, что значение выражения $L_t(f)$ (без последнего члена $f^{(1)} u_{(t)}$) может быть вычислено при известных $u_{(1)}, u_{(2)}, \ldots, u_{(t-1)}$ по алгоритму **A1**; таким образом, грубо говоря, искомый алгоритм имеет структуру вида

$$
\bigcup_{t=1}^{n} \left\{ \mathbf{A1}(t); \qquad u_{(t)} \right\}
$$

Точная запись этого алгоритма выглядит следующим образом.

Алгоритм A2

вход $(n;\ b\,[1:n])$; рабочие массивы $a\,[1:n],\ c\,[0:n-1]$;
if $b_1 = 0$ **then goto** M; $a_1 := 1/b_1$

$$\overset{n}{\underset{t=2}{\Bigg\{}} \overset{t-1}{\underset{i=0}{\Bigg\{}} \overset{k=i}{\underset{1}{\Bigg\{}} c_k := \frac{1}{k}\sum_{j=1}^{k} a_j * c_{k-j} \Bigg\}\ c_0 := b_{t-i} \Bigg\}\ a_t := -a_1 * \sum_{j=1}^{t-1} a_j * c_{t-j} \Bigg\}$$

$$\overset{n}{\underset{t=1}{\Bigg\{}} b_t := \frac{a_t}{t} \Bigg\} \quad \mathrm{M}: \qquad (\mathrm{b})$$

Пояснение. На входе $b_j = f^{(j)}$, на выходе, если $b_1 = 0$, то обратной функции не существует (в этом случае массив b тот же, что на входе), если $b_1 \neq 0$, то $b_j = u_{(j)}\ \ j = 1, 2, \ldots, n;\ \ n \geq 1$, алгоритм требует $\sim n^4/24$ операций, при $n = 10$ это ~ 450 операций.

Замечание. Для целей данного пункта можно было бы построить алгоритм с оценкой числа операций $n^3/6$, базирующийся на измененной схеме алгоритма **A1**, работающий не "ходом назад", а "ходом вперед". Однако такой алгоритм оказывается менее устойчивым к вычислительным погрешностям (на подобных вопросах мы в настоящем Сообщении не останавливаемся), и, кроме того, он не обобщается на более общую задачу нахождения производных неявной функции. По этим причинам мы отдали предпочтение описанному алгоритму **A2** (ожидаемые значения n все же не очень велики, $n\ \sim\ 10$).

4. Частные производные сложной функции двух переменных

Факторизованные частные производные двух переменных $\Phi(x, u)$ определяются аналогичным образом:

$$\Phi_{kt} := \frac{1}{k!t!}\frac{\partial^{k+t}\Phi}{\partial x^k \partial u^t}\ , \quad k, t, = 0, 1, \ldots \quad ,$$

при этом сумма $k + t$ называется порядком производной. Эти производные удовлетворяют соотношениям линейности вида (1), а для произведения двух функций имеет место формула свертки, аналогичная (5)

$$(\Phi F)_{kt} = \sum_{i=0}^{k}\sum_{j=0}^{t} \Phi_{ij} * F_{k-i,t-j} \quad . \qquad (12)$$

Как и в п.1, можно, исходя из (12), построить простые рекуррентные алгоритмы вычисления частных производных специальных функций $\Psi(x,u) = 1/\Phi(x,u)$ и $\Psi(x,u) = exp(\Phi(x,u))$; ввиду полной аналогии, мы не будем на этом останавливаться, а обратимся сразу к общему случаю сложной функции $\Psi(x,u) = f(\Phi(x,u))$, где $f(z)$ – произвольная (дифференцируемая)функция **одного** переменного. Подчеркнем, что вместе с операцией сложения и умножения этого достаточно для включения в алгебру дифференцирования подавляющего большинства функций двух переменных, используемых на практике; например, функцию $\Phi_1(x,u)^{\Phi_2(x,u)}$ можно представить в виде $\exp(\Phi_2 \cdot \ln \Phi_1)$, т. е. в виде суперпозиции двух функций одного переменного (плюс операция умножения: имея Φ_1, Φ_2 строим последовательно $\Psi_1 := \ln \Phi_1,\ \Psi_2 := \Phi_2 \cdot \Psi_1,\ \Psi_3 := \exp(\Psi_2)$.

Итак, обратимся к задаче вычисления всех факторизованных производных Ψ_{kt} порядков $\leq n$ $(k+t \leq n)$ сложной функции $\Psi(x,u) = f(z)$, $z = \Phi(x,u)$ в данной точке (x_0, u_0), где функции f и Φ заданы; исходной информацией в этой задаче будут значения производных Φ_{kt}, $0 \leq k+t \leq n$ функции Φ в точке (x_0, u_0) и (обычных) производных $f^{(0)}, f', f'', \ldots, f^{(n)}$ в точке $z_0 = \Phi(x_0, u_0)$.

Последуем схеме рассуждений, которая привела в п.2 к алгоритму **A1**; зафиксируем функцию $\Phi(x,u)$, и для произвольной функции $g(z)$ положим по определению

$$L_{kt}(g) := \frac{1}{k!t!}\frac{\partial^{k+t} g}{\partial x^k \partial u^t}, \quad g = g(z),\ z = \Phi(x,u),\ k,t = 0,1,2,\ldots \quad .$$

Наша задача – построить алгоритм вычисления "пирамиды" производных

$$\begin{pmatrix} L_{00} \\ L_{10}\ L_{01} \\ L_{20}\ L_{11}\ L_{02} \\ \ldots\ldots\ldots\ldots \\ L_{n0}\ L_{n-1,1} \ldots L_{0n} \end{pmatrix}_f \tag{13}$$

в точке (x_0, u_0). Применим к равенству

$$L_{10}(g) = g' \cdot \frac{\partial \Phi}{\partial x} = g' \Phi_{10}$$

формулу (12), получим при $k \geq 1$ (аналогично (9))

$$\begin{aligned} L_{kt}(g) &= \frac{1}{k} L_{k-1,t}\big(L_{10}(g)\big) = \frac{1}{k} L_{k-1,t}(g' \cdot \Phi_{10}) = \\ &= \frac{1}{k} \sum_{i=0}^{k-1} \sum_{j=0}^{t} \big(\Phi_{10}\big)_{ij} \cdot L_{k-1-i,t-j}(g') = \\ &= \frac{1}{k} \sum_{i=1}^{k} \sum_{j=0}^{t} i\Phi_{ij} \cdot L_{k-i,t-j}(g') \quad . \end{aligned}$$

Подставляя сюда в качестве g функцию $f^{(m)}$, получим

$$L_{kt}\big(f^{(m)}\big) = \frac{1}{k} \sum_{i=1}^{k} \sum_{j=0}^{t} i\Phi_{ij} \cdot L_{k-i,t-j}\big(f^{(m+1)}\big) \quad k = 1, 2, \ldots, \quad t = 0, 1, \ldots \quad . \tag{14}$$

При $k = 0$ мы имеем дело с производными только по одному аргументу (по u), и поэтому мы можем воспользоваться формулой (10), относящейся к функции одного переменного; это дает

$$\begin{cases} L_{0t}\big(f^{(m)}\big) = \dfrac{1}{t} \displaystyle\sum_{j=1}^{t} j\Phi_{0j} \cdot L_{0,t-j}\big(f^{(m+1)}\big) \quad t = 1, 2, \ldots \\ L_{00}\big(f^{(m)}\big) = f^{(m)} \quad . \end{cases} \tag{14}$$

формулы (14) показывают, что для вычисления пирамиды (13) порядка n достаточно иметь пирамиду $(n-1)$–го порядка

$$\begin{pmatrix} L_{00} \\ L_{10}\ L_{01} \\ \ldots\ldots\ldots\ldots \\ L_{n-1,0}\ L_{n-2,1} \ldots L_{0,n-1} \end{pmatrix}_{f'}$$

для этого, в свою очередь, достаточно иметь пирамиду $(n-2)$–го порядка для функции $f^{(2)}$ и т. д. ; заканчивается все пирамидой нулевого порядка $L_{00}\big(f^{(n)}\big) = f^{(n)}$. Это приводит к "двумерному" алгоритму совершенно аналогичному по идее "одномерному" алгоритму **A1**; этот алгоритм, конечно, более громоздок, но, оказывается, для его реализации достаточно иметь рабочее поле, состоящее лишь из одной квадратной матрицы и одного вектора порядка $(n+1)$ (на этом поле располагается и исходная информация).

Алгоритм А3.
вход $(n;\ E[0:n,\ 0:n],\ b[0:n])$

$$E_{00} := b_n \quad \overset{n}{\underset{p=1}{\Bigg\{}} m := n-p \quad b_{m+1} := p * E_{pp} \quad \overset{p}{\underset{j=1}{\Bigg\{}} E_{p-j,p} := j * E_{p,p-j} \Bigg\}$$

$$\overset{t=p-1}{\underset{0}{\Bigg\{}} \overset{k=p-t}{\underset{1}{\Bigg\{}} q := k+t \quad E_{qt} := \frac{1}{k} \sum_{j=0}^{t} \sum_{i=1+j}^{k+j} E_{ji} * E_{q-i,t-j} \Bigg\}$$

$$q := t+1 \quad E_{qq} := \frac{1}{q} \sum_{j=0}^{t} b_{n-t+j} * E_{jj} \Bigg\} \quad E_{00} := b_m \Bigg\} \qquad (E)$$

Пояснение. На входе значения $f^{(j)},\ j = 0, 1, \ldots, n$ находятся в ячейках b_j, пирамида значений $\Phi_{kt},\ 0 \le k+t \le n$ занимает нижнюю половину (под главной диагональю и на ней) матрицы E, именно значение Φ_{kt} находится в ячейке $E_{k+t,t}$ (т. е. пирамида строится по типу (13)); содержимое верхней половины матрицы E и ячейки E_{00} безразлично – это рабочее поле алгоритма. На выходе значения $L_{kt}(f)$ (т. е. пирамида (13)) расположены в матрице E аналогично, это удобно для многократного обращения к алгоритму: при последующих обращениях надо подготавливать только массив b (сравни с пояснением 3 к алгоритму **А1**). Допустимо $n \ge 1$; алгоритм требует $\sim\ n^5/120$ операций, при $n = 10$ это $\sim\ 10^3$ операций.

5. Полные производные функции двух переменных

Пусть заданы функция $u(x)$ и функция двух переменных $\Phi(x, u)$. Как и в п.2, зафиксировав функцию $u(x)$, будем обозначать через $L_k(G)$ полную $k-$ю факторизованную производную произвольной функции двух переменных $G(x, u)$, т. е.

$$L_k(G) := \frac{1}{k!} \frac{d^k G}{dx^k}\ , \quad G = G(x, u), \quad u = u(x),\ k = 0, 1, \ldots \quad . \qquad (15)$$

Задача настоящего пункта – построение алгоритма вычисления $(L_0, L_1, \ldots, L_n)\ \Phi|_{x=x_0}$ при известных $u', u'', \ldots, u^{(n)}\ |_{x_0}$ и частных производных (обычных, обозначаемых верхними индексами)

$$\Phi^{pq}|_{(x_0,u_0)}, \quad u_0 = u(x_0), \quad 0 \le p+q \le n \quad .$$

Для произвольной функции $G(x, u)$ имеем

$$L_1(G) = \frac{\partial G}{\partial x} + \frac{\partial G}{\partial u} u' = G^{10} + G^{01} \cdot u_{(1)} \quad ,$$

поэтому на основании (5)б (6) запишем аналогично (9) при $k \geq 1$

$$\begin{aligned} L_k(G) &= \frac{1}{k} L_{k-1}\big(L_1(G)\big) = \frac{1}{k} L_{k-1}\big(G^{10} + G^{01} \cdot u_{(1)}\big) = \\ &= \frac{1}{k}\big(L_{k-1}(G^{10}) + \sum_{j=1}^{k} j u_{(j)} L_{k-j}(G^{01})\big) \quad . \end{aligned}$$

Подставляя сюда в качестве G функцию Φ^{pq}, получим

$$\left\{ \begin{array}{lcl} L_k\big(\Phi^{pq}\big) &=& \frac{1}{k}\big(L_{k-1}(\Phi^{p+1,q}) + \sum_{j=1}^{k} j u_{(j)} L_{k-j}(\Phi^{p,q+1})\big) \; k \geq 1 \\ L_0\big(\Phi^{pq}\big) &=& \Phi^{pq} \quad . \end{array} \right. \tag{16}$$

Из этих рекуррентных соотношений следует, что для нахождения $(L_0, L_1, \ldots, L_n)$ от функции Φ достаточно знать $(L_0, L_1, \ldots, L_{n-1})$ от производных первого порядка $(p+q=1)$, для этого, в свою очередь, достаточно иметь $(L_0, L_1, \ldots, L_{n-2})$ от производных второго порядка $(p+q=2)$ и т. д. вплоть до L_0 от производных $n-$го порядка. Так же, как и в предыдущем пункте, эта ситуация вполне аналогична “одномерной” ситуации п.2 и приводит к следующему алгоритму.

Алгоритм А4

вход $(n;\ a\,[0:n],\ E\,[0:n,\ 0:n])$

$$\overset{n}{\underset{i=1}{\Big\{}} a_i := i * a_i \quad m := n - i \quad \overset{n}{\underset{q=0}{\Big\{}} \ \overset{k=i}{\underset{1}{\Big\{}} p := k + m$$

$$E_{pq} := \frac{1}{k}\big(E_{pq} + \sum_{j=1}^{k} a_j * E_{p+1-j,q+1}\big)\Big\}\Big\}\Big\}$$

$$\overset{n}{\underset{j=0}{\Big\{}} a_j := E_{j0} \Big\} \qquad (a)$$

Пояснение. На входе $a_j = u_{(j)},\ \ j = 0, 1, \ldots, n$, значения Φ^{pq} расположены как и в алгоритме **А3**, а именно, Φ^{pq} находится в ячейке $E_{p+q,q}$ матрицы $E,\ 0 \le p + q \le n$. На выходе $a_j = L_j(\Phi)$, $j = 0, 1, \ldots, n$; более полная информация содержится в матрице E, именно, $E_{p+q,q} = L_p(\Phi^{0q})$, $0 \le p + q \le n$. Верхняя половина матрицы E (над главной диагональю) алгоритмом не используется; допустимо $n \ge 0$, алгоритм требует $\sim\ n^4/24$ операций, при $n = 10$ это $\sim\ 450$ операций.

6. Производные неявной функции

Пусть заданы функция $\Phi(x, u)$ и точка (x_0, u_0), удовлетворяющая условию $\Phi(x_0, u_0) = 0$; неявной функцией $u(x)$ мы называем решение уравнения $\Phi(x, u) = 0$, определенное и непрерывное в окрестности точки $x = x_0$ и такое, что $u(x_0) = u_0$. Цель этого пункта – дать алгоритм вычисления $u_{(0)}, u_{(1)}, \ldots, u_{(n)}\ |_{x_0}$ при известных частных производных $\Phi^{pq}|_{(x_0,u_0)}$, $1 \le p + q \le n$ (подчеркнем, что значение u_0 не находится, оно считается известным); эта задача является обобщением задачи п.3, которая может быть представлена как задача о неявной функции, если положить $\Phi(x, u) = f(u) - x$.

Пусть $u(x)$ – неявная функция, и операторы L_k определены в (15), тогда $L_k(\Phi) = 0,\ \ k = 1, 2, \ldots, n$. Из рекуррентных соотношений (16) легко усмотреть, что t−я производная $u_{(t)}$ появляется впервые в выражении для $L_t(\Phi)$ и при этом с коэффициентом $\Phi^{01} = \dfrac{\partial \Phi}{\partial u}|_{(x_0,u_0)}$. Для существования (однозначной) неявной функции в окрестности точки x_0 необходимо и достаточно, чтобы $\Phi^{01} \ne 0$; поэтому значение $u_{(1)}$ может быть найдено из условия $L_1(\Phi) = 0$, после этого, зная $u_{(1)}$, можно найти $u_{(2)}$

из условия $L_2(\Phi) = 0$ и т. д. . Вообще, $u_{(t)}$ может быть найдено из условия $L_t(\Phi) = 0$, при этом значение выражения для $L_t(\Phi)$ без последнего члена $\Phi^{01} \cdot u_{(t)}$ может быть вычислено при известных $u_{(1)}, u_{(2)}, \ldots, u_{(t-1)}$ по алгоритму **A4**. Рассматриваемая ситуация вполне аналогична п.3 и приводит к следующему алгоритму.

Алгоритм A5

вход $(n;\ E\,[0:n, 0:n])$ **if** $E_{11} = 0$ **then goto** $M1$;

$$\begin{array}{l}
\overset{n}{\underset{t=1}{\Big\{}} E_{0,t-1} := 0 \qquad \overset{t}{\underset{i=0}{\Big\{}} m := t - i \\
\overset{m}{\underset{q=0}{\Big\{}} \overset{k=i}{\underset{1}{\Big\{}} p := k + m \quad E_{qp} := \frac{1}{k}\big(E_{qp} + \sum_{j=0}^{k-1} E_{0j} * E_{q+1,p-j} \\
\textbf{if}\ \ q = 0\ \ \textbf{then goto}\ \ M2 \Big\} \quad E_{qm} := E_{mq} \quad M2: \Big\}\Big\} \\
E_{0,t-1} := -\frac{t * E_{0t}}{E_{11}} \Big\} \qquad \overset{t=n}{\underset{1}{\Big\{}} E_{0t} := \frac{E_{0,t-1}}{t} \Big\}
\end{array}$$

$M1:$ выход (E)

Пояснение. На входе матрица E содержит частные производные Φ^{pq}, $1 \le p + q \le n$, как это указано в пояснении к алгоритму **A4**. На выходе, если $E_{11} = 0$, то неявной функции не существует, в этом случае матрица E та же, что на входе; если $E_{11} \ne 0$, то $u_{(j)}$ находятся в ячейках E_{0j}, $j = 1, 2, \ldots, n$. Допустимо $n \ge 1$, рабочее поле матрицы E используется полностью; алгоритм требует $\sim\ n^5/120$ операций, при $n = 10$ это $\sim\ 10^3$ операций.

7. Замечания к использованию алгоритмов

Для приложений часто бывает удобнее пользоваться однородными производными (обозначим их индексом в квадратных скобках)

$$y^{[k]} := \gamma^k y^{(k)}, \quad y[k] := \gamma^k y(k), \quad k = 0, 1, 2, \ldots$$

с некоторым параметром γ, имеющим размерность аргумента функции $y(x)$, $\gamma \neq 0$. Например, γ может быть единицей измерения аргумента, или масштабным множителем, выбранным так, чтобы величины $y^{[k]}$ (или $y_{[k]}$) не очень быстро росли или убывали с номером k. Так, при использовании формулы (3) удобно взять $\gamma = -x_0$, а для ряда (4) $\gamma = x^{-1}$.

Приведенные алгоритмы **A1** – **A5** можно было бы написать в терминах однородных производных (они при этом практически не изменились бы), но чтобы не загромождать изложение, мы решили ограничиться обычными производными (что является частным случаем однородных, получающимся при $\gamma = 1$); при желании необходимые изменения легко внести. Поэтому, использование алгоритмов **A1** – **A5** (в том виде, как они приведены) требует определенной осторожности. Так, например, производные функции $y(t) = \sin \omega t$ растут как ω^k, что при больших ω может привести в ЭВМ к переполнению даже при небольших k (и, наоборот, если ω мало, то ω^k быстро обратится в машинный нуль). Аналогично производные функции $u(x)$, обратной к функции $f(u) = A \sin u$, растут (или убывают) как A^{-k}, т. к. в данном случае $u(x) = \arcsin \frac{x}{A}$. Поэтому перед использованием алгоритмов **A1** – **A5** надо привести задачу к безразмерным аргументам.

Литература

1. Эрдейи А. Асимптотические разложения. – М.: ГИФМЛ, 1962.

2. Moore R.E. Interval analysis. – Prentice-Hall: Englwood Cliffs, N.Y., 1966.

3. Barton D., Willers I.M., Zahar R.V.M. Taylor series methods for ordinary differential equations – an evaluation. – In: Math. Software. New York: Acad. Press, 1971, p. 369–390.

4. Barton D., Willers I.M., Zahar R.V.M. The automatic solution of system of ordinary differential equations by the method of Taylor series. – Comput. J., v. 14. N 3, 1971, p.243–248.

5. Гофен А.М. Быстрое разложение в ряд Тейлора и решение задачи Коши. – ЖВМ и МФ, 1982, т. 22, N 5, с. 1094–1108.

6. Никуличев Ю.В. Об одном способе построения программ аналитического дифференцирования функций на ЭВМ. – В кн. Оптимизация динамических систем. – Новосибирск: Институт теоретической и прикладной механики, 1979, с. 47–59.

7. Todorov P.G. New explicit formulas for n-th derivative of composite functions. – Pacific Journal of mathematics, v. 92, N1, 1981, p. 217–236.

Приложение

Прилагаемые процедуры записаны на языке Алгол–68 в версии, принятой в ЦЭМИ АН СССР по состоянию на май 1982 года. Все процедуры прошли тестовую проверку на ЭВМ ЕС–1022, тестовая информация приводится.

Процедура dif 1 – вычисление факторизованных производных сложной функции $\Psi(x) = f(u),\ u = u(x)$ в данной точке x_0.

```
1 .proc dif 1=(.ref[ ].real A,B) .void :
2 .begin .int i,j,k,m,p; .real r; .int n=.upb A;
3 .for  i .to n .do A[i]:=i*A[i]; m:=n–i;
4 .for  k .from i .by –1 .to 1 .do r:=0; p:=k+m;
5 .for  j .to k .do r:=r+A[j]*B[p+1–j]  .od; B[p]:=r/k  .od .od;
6 .for  j .from 0 .to n .do A[j]:=B[j]  .od .end;
```

Массивы A, B имеют в основной программе описание [0:n].**real** A, B; текст процедуры соответствует алгоритму **A1** Сообщения. Тестовый пример

$$n = 4,\ u(x) = \pi x^4,\ f(u) = \sin u,\ x_0 = \frac{1}{\sqrt[4]{3}} = 0.7598 \quad .$$

На входе – факторизованные производные функции u:
$u_{(0)},\ u_{(1)},\ u_{(2)},\ u_{(3)},\ u_{(4)}|_{x=x_0}$

$$A = (1.0472,\ 5.5128,\ 10.833,\ 9.5484,\ 3.1416) \quad ,$$

обычные производные функции $f:\ f^{(0)},\ f^{(1)},\ f^{(2)},\ f^{(3)},\ f^{(4)}|_{u=u_0}$

$$B = (0.8660,\ 0.5 \quad ,\ -0.8660,\ -0.5 \quad ,\ 0.8660) \quad .$$

На выходе – факторизованные производные $\Psi_{(j)},\ j = 0, 1, 2, 3, 4$

$$A = (0.8660,\ 2.7560,\ -7.7180,\ -61.143,\ -144.65) \quad .$$

Процедура dif 2 – вычисление факторизованных производных обратной функции; прямая функция – $x = f(u)$ с заданным значением u_0 независимого аргумента u, обратная функция – $u = u(x) := f^{-1}(x)$, производные которой вычисляются (если $f'(u_0) \neq 0$).

```
1 .proc dif2=(.ref[ ].real B) .void : .begin .int i,j,k,t;
2 .int n= .upb B; [1:n].real A; [0:n-1].real C; .real r;
3 .if .abs B[1]< 0.1E-11 .then  goto M .fi; A[1]:=1/B[1];
4 .for  t .from 2 .to n .do .for  i .from 0 .to t-1 .do
5 .for  k .from i .by -1 .to 1 .do r:=0; .for  j .to k .do
```

```
6  r:=r+A[j]*C[k-j] .od; C[k]:=r/k .od; C[0]:=B[t-i] .od;
7  r:=0; .for  j .to t-1 .do r:=r-A[j]*C[t-j]  .od; A[t]:=A[1]*r  .od;
8 .for  t .to n .do B[t]:=A[t]/t  .od; M: .skip  .end;
```

Массив B имеет описание [1:n].**real** B; текст процедуры соответствует алгоритму **A2** Сообщения. Тестовый пример

$$n = 4, \quad f(u) = \sin u, \quad u_0 = -\frac{\pi}{4}$$

и, следовательно,

$$u(x) = \arcsin x, \quad x_0 = -\frac{1}{\sqrt{2}} \quad .$$

На входе – обычные производные функции f : $f^{(1)},\ f^{(2)},\ f^{(3)},\ f^{(4)}|_{u=u_0}$

$$B = (0.7071,\ 0.7071,\ -0.7071,\ -0.7071) \quad .$$

На выходе – факторизованные производные u:
$u_{(1)},\ u_{(2)},\ u_{(3)},\ u_{(4)}|_{x=x_0}$.

$$B = (1.4142,\ -1,\ 1.8856,\ -4) \quad .$$

Процедура dif 3 – вычисление факторизованных частных производных

$$\Psi_{kt} := \frac{1}{k!t!}\frac{\partial^{k+t}\Phi}{\partial x^k \partial u^t}\ , \quad 0 \le k+t \le n$$

сложной функции $\Psi(x,u) = f(z),\ z = \Phi(x,u)$ в данной точке (x_0, u_0).

```
1 .proc dif3=(.ref[ , ].real E, .ref[ ].real B) .void : .begin
2 .real r; .int i,j,k,m,p,q,t; .int n=.upb B; E[0,0]:=B[n];
3 .for  p .to n .do m:=n-p; B[m+1]:=p*E[p,p]; .for  j .to p .do
4  E[p-j,p]:=E[p,p-j]*j  .od; .for  t .from p-1 .by -1 .to 0 .do
5 .for  k .from p-t .by -1 .to 1 .do q:=k+t; r:=0;
6 .for  j .from 0 .to t .do .for  i .from 1+j .to k+j .do
7  r:=r+E[j,i]*E[q-i,t-j]  .od .od; E[q,t]:=r/k  .od; q:=t+1; r:=0;
8 .for  j .from 0 .to t .do r:=r+B[n-t+j]*E[j,j]  .od;
9  E[q,q]:=r/q  .od; E[0,0]:=B[m]  .od .end;
```

Входные массивы имеют описания [0:n,0:n].**real** E, [0:n].**real** B; текст процедуры соответствует алгоритму **A2** Сообщения. Тестовый пример

$$n = 3,\ f(z) = 2\sin z,\ \Phi(x,u) = \frac{1}{x^2 u},\ (x_0, u_0) = (2, \frac{3}{4\pi}) \quad .$$

На входе – факторизованные производные $\Phi_{kt},\ 0 \le k+t \le 3$

$$E = \begin{pmatrix} 1.0472 & & & \\ -1.0472 & -4.3865 & & \\ -0.7854 & 4.3865 & 18.374 & \\ -0.5236 & -3.2899 & -18.374 & -76.965 \end{pmatrix}, E_{k+t,t} = \Phi_{kt} \quad ;$$

обычные производные $f^{(j)},\ j = 0, 1, 2, 3$

$$B = (1.7321,\ 1,\ -1.7321,\ -1) \quad .$$

На выходе – факторизованные производные $\Psi_{kt},\ 0 \le k+t \le 3$

$$E = \begin{pmatrix} 1.7321 & & & \\ -1.0472 & -4.3865 & & \\ -0.1643 & -3.5697 & 1.7106 & \\ 1.0923 & 13.039 & 58.354 & 76.701 \end{pmatrix}, E_{k+t,t} = \Psi_{kt} \quad .$$

Процедура dif 4 – вычисление полных факторизованных производных

$$\Psi_{(k)} := \frac{1}{k!}\frac{\mathrm{d}^k \Psi}{\mathrm{d}x^k}\ , \quad k = 0, 1, \ldots, n$$

сложной функции $\Psi(x) = \Phi(x, u),\ u = u(x)$ в данной точке x_0.

```
1 .proc dif4=(.ref[ ].real A, .ref[ , ].real E) .void :
2 .begin .real r; .int i,j,k,t,p,m; .int n=.upb A;
3 .for  i .to n .do A[i]:=A[i]*i; m:=n-i; .for  t .from 0 .to m .do
4 .for  k .from i .by -1 .to 1 .do p:=k+m; r:=E[p,t]; .for  j .to k .do
5  r:=r+A[j]*E[p+1-j,t+1]  .od; E[p,t]:=r/k  .od .od .od;
6 .for  j .from 0 .to n .do A[j]:=E[j,0]  .od .end;
```

Входные массивы имеют описания [0:n].**real** A, [0:n,0:n].**real** E; текст процедуры соответствует алгоритму **A4** Сообщения. Тестовый пример

$$n = 3,\ \Phi(x, u) = \sin(xu) - \sin(1.2 \cdot e^{1.2}),\ u(x) = e^x,\ x_0 = 1.2 \quad .$$

На входе – факторизованные производные $u_{(j)},\ j = 0, 1, 2, 3$

$$A = (3.3201,\ 3.3201,\ 1.6601,\ 0.5534) \quad ,$$

обычные производные $\Phi^{kt},\ 0 \le k+t \le 3$

$$E = \begin{pmatrix} 0 & & & \\ -2.2098 & -0.7987 & & \\ 8.2270 & 2.3080 & 1.0747 & \\ 24.3580 & 13.7600 & 4.9733 & 1.1501 \end{pmatrix}, E_{k+t,t} = \Phi_{kt} \quad .$$

На выходе – факторизованные производные $\Psi_{(k)},\ k = 0, 1, 2, 3$

$$A = (0,\ -4.8615,\ 16.374,\ 70.640) \quad .$$

Процедура dif 5 – вычисление факторизованных производных неявной функции $u(x)$, заданной уравнением $\Phi(x, u) = 0$, в данной точке x_0; значение $u_0 = u(x_0)$ должно быть вычислено вне процедуры. Неявная функция определена только, если

$$\frac{\partial \Phi}{\partial u}|_{(x_0,u_0)} \neq 0 \quad .$$

```
1 .proc dif5=(.ref[ , ].real E) .void : .begin .real r;
2 .int n=.upb E; .int i,j,k,m,t,p,q; .if .abs E[1,1]<0.1E-11
3 .then  goto M1 .fi; .for  t .to n .do E[0,t-1]:=0;
4 .for  i .from 0 .to t .do m:=t-i; .for  q .from 0 .to m .do
5 .for  k .from i .by -1 .to 1 .do p:=k+m; r:=E[q,p];
6 .for  j .from 0 .to k-1 .do r:=r+E[0,j]*E[q+1,p-j]  .od;
7 E[q,p]:=r/k; .if q=0 .then  goto M2 .fi .od; E[q,m]:=E[m,q];
8 M2: .skip  .od .od; E[0,t-1]:=-t*E[0,t]/E[1,1]  .od;
9 .for  t .from n .by -1 .to 1 .do E[0,t]:=E[0,t-1]/t  .od;
10 M1: .skip  .end;
```

Входной массив имеет описание [0:n,0:n].**real** E; текст процедуры соответствует алгоритму **A5** Сообщения. Тестовый пример

$$n = 3,\ \Phi(x, u) = \sin(u + x^3) - \frac{1}{2},\ x_0 = 1,\ u_0 = \frac{\pi}{6} - 1\ ;$$

соответствующая неявная функция будет

$$u(x) = \frac{\pi}{6} - x^3 \quad .$$

На входе – обычные производные $\Phi^{kt},\ 0 \leq k + t \leq 3$

$$E = \begin{pmatrix} 0 & & & \\ 2.5981 & 0.8660 & & \\ 0.6962 & -1.5000 & -0.5000 & \\ -45.1870 & -10.7940 & -2.5981 & -0.8660 \end{pmatrix},\ E_{k+t,t} = \Phi^{kt} \quad .$$

На выходе

$$E = \begin{pmatrix} - & -3 & -3 & -1 \\ - & & & \\ - & & & \\ - & & & \end{pmatrix} \quad ,$$

здесь элементы верхней строки дают искомые значения $u_{(j)} = E_{0,j},\ j = 1, 2, 3.$

Процедура dif 6 – нахождение коэффициентов отрезка ряда Тейлора

$$u(x) = u_0 + c_1(x - x_0) + c_2(x - x_0)^2 + \ldots + c_n(x - x_0)^n$$

решения задачи Коши для дифференциального уравнения

$$u' = \Phi(x, u), \quad u(x_0) = u_0 \quad .$$

здесь $c_k = u_{(k)}$ – $k-$я факторизованная производная функции u в точке x_0; эта задача кратко описана в преамбуле Сообщения.

```
1 .proc dif6=(.ref[ ].real A, .ref[ , ].real E) .void :
2 .begin .real r; .int i,j,k,t,p,,m,q; .int n=.upb A;
3 .for  t .from 0 .to n-1 .do .for  j .to t .do
4 .for  i .from 0 .to j-1 .do E[i,j]:=E[j,i]  .od .od;
5 .for  i .to t .do m:=t-i; .for  q .from 0 .to m .do
6 .for  k .from i .by -1 .to 1 .do p:=k+m; r:=E[q,p];
7 .for  j .to k .do r:=r+A[j]*E[q+1,p+1-j]  .od;
8  E[q,p]:=r/k  .od .od .od; A[t+1]:=E[0,t]  .od;
9 .for  j .to n .do A[j]:=A[j]/j  .od .end;
```

Входные массивы имеют описания [0:n].**real** A, [0:n-1,0:n-1].**real** E; тестовый пример

$$n = 3,\ \Phi(x, u) = -\frac{1}{2x^2u},\ x_0 = 1,\ u_0 = 1 \quad ;$$

решением задачи Коши является функция $u(x) = 1/\sqrt{x}$. На входе – обычные производные $\Phi^{kt},\ 0 \le k + t \le 2$

$$E = \begin{pmatrix} -0.5 & & \\ 1 & -0.5 & \\ -3 & -1 & -1 \end{pmatrix},\ E_{k+t,t} = \Phi^{kt} \quad .$$

Массив A на входе не задается (он служит адресом засылки результата), кроме ячейки A_0, куда целесообразно заслать значение u_0, в примере полагаем $A_0 = 1$.
На выходе – факторизованные производные $u_{(k)}|_{x_0},\ k = 0, 1, 2, 3$

$$A = (1,\ -0.54,\ 0.375,\ -0.3125)$$

(если A_0 не было задано перед обращением к процедуре, то оно не определено).

* * *