О.В.Бартеньев

ФОРТРАН ДЛЯ ПРОФЕССИОНАЛОВ. Математическая библиотека IMSL. Выпуск 3

Излагаются средства математической библиотеки IMSL, входящей в состав профессиональных версий Фортрана фирм Microsoft и Compaq, позволяющие строить сплайны, выполнять интегрирование и дифференцирование, решать дифференциальные уравнения. Представляемый материал иллюстрируется большим числом примеров.

Предназначено для научных работников, инженеров, преподавателей, студентов и аспирантов вузов.

Содержание

Предметный указатель

1. ИНТЕРПОЛЯЦИЯ И АППРОКСИМАЦИЯ

1.1. ВВЕДЕНИЕ

Процедуры библиотеки IMSL 77, формирующие сплайны, осуществляют либо интерполяцию, либо аппроксимацию (приближение, или сглаживание) данных. Также библиотека включает вспомогательные, относящиеся к сплайнам процедуры, выполняющие их оценку, интегрирование, вычисление производных и преобразование из одного представления в другое.

Процедуры преимущественно ориентированы на поиск кубических интерполяционных сплайнов и B-сплайнов. Процедуры первого вида начинаются с букв CS, а второго - с букв BS. Большинство процедур основаны на программах, приведенных в [7].

При изложении материала с целью экономии места опускаются процедуры второго уровня, а также данные о размерах автоматически выделяемой памяти.

Отдельно, в следующей главе, рассматриваются средства библиотеки IMSL 90, в том числе функции spline_fitting и surface_fitting, применяемые для построения по методу наименьших квадратов B-сплайнов и имеющие дополнительные по сравнению с аналогичными процедурами библиотеки IMSL 77 возможности.

1.2. ОБЗОР ПРОЦЕДУР

1.2.1. ИНТЕРПОЛЯЦИЯ МНОГОЧЛЕНАМИ

Кусочно-многочленная (КМ) функция одной независимой переменной $p(x)$ - это кривая, составленная из частей (кусков) отдельных многочленов. КМ-функция задается последовательностью точек разрыва $\xi \in R^n$, порядком k (степенью k - 1) ее отдельных кусков и $k \times (n$ - 1)-матрицей C ее локальных (относящихся к отдельным кускам) полиномиальных коэффициентов. Иными словами, КМ-функция - это

$$p(x) = \sum_{j=1}^{k} c_{ji} \frac{(x-\xi_i)^{j-1}}{(j-1)!}, \quad \xi_i \le x < \xi_{i+1}.$$

Последовательность точек разрыва ξ должна быть возрастающей.

Приведенное представление сплайна сложнее, чем нужно, когда известно, что КМ-функция гладкая. Например, если известно, что p - непрерывная функция, то можно вычислить $c_{1,i+1}$ по известным c_{ij} следующим образом:

$$c_{1,j+1} = p(\xi_{i+1}) = c_{1,j} + c_{2,j}\Delta\xi_i + \ldots + c_{ki}\frac{(\Delta\xi_i)^{k-1}}{(k-1)!},$$

где $\Delta\xi_i = \xi_{i+1} - \xi_i$. Для гладкой КМ-функции предпочтительнее использовать B-сплайны.

1.2.2. B-СПЛАЙНЫ

B-сплайны обеспечивают удобный для применения механизм построения гладких КМ-функций некоторого класса. Класс функций характеризуется последовательностью точек разрыва, порядком сплайна и заданной степенью гладкости в каждой внутренней точке разрыва. Соответствующий B-сплайн порядка k находится по заданной последовательности $t \in \mathbf{R}^M$. Последовательность узлов должна удовлетворять следующим требованиям:

- если КМ-функция должна иметь в точке ξ_i производные j-го порядка включительно, то величина ξ должна входить 1 раз в последовательность узлов $k - j$;
- ξ_1 и ξ_n являются концевыми точками рассматриваемого интервала; первые k узлов последовательности должны равняться ξ_1, а k последних - ξ_n. Это обеспечит непрерывность справа B-сплайна вблизи ξ_1 и непрерывность слева вблизи ξ_n.

После удовлетворения этих требований генерируется последовательность узлов - вектор t размера M, а затем вычисляются $m = M - k$ B-сплайнов $B_i, \ldots, B_m$ порядка k, составляющих КМ-функцию на заданном интервале и имеющую на нем заданную степень гладкости. Каждая КМ-функция имеет уникальное представление в виде линейной комбинации B-сплайнов:

$$p = a_1B_1 + a_2B_2 + \ldots + a_mB_m .$$

B-сплайн B_i - это неотрицательная функция, отличная от нуля только на последовательности интервалов $[t_i, t_{i+k}]$. Такая последовательность интервалов называется *носителем* B-сплайна. Причем любая иная КМ-функция, входящая в тот же, что и B-сплайн, класс, имеет больший, чем B-сплайн B_i, носитель. Это обстоятельство делает B-сплайны весьма привлекательными для использования в качестве базисных функций: влияние конкретного B-сплайна распространяется лишь на несколько интервалов.

Обозначение B_{ikt} используется в дальнейшем для отражения зависимости B-сплайна от его параметров. Это обозначение говорит о том, что B-сплайн B, имеет порядок k на последовательности узлов t.

1.2.3. КУБИЧЕСКИЕ СПЛАЙНЫ

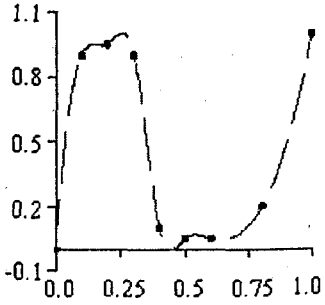
Кубические сплайны - это гладкие, обладающие C^1- или C^2-непрерывностью КМ-функции 4-го порядка (третьей степени). Поскольку кубические сплайны используются в задачах интерполяции и аппроксимации достаточно интенсивно, IMSL содержит большое число процедур для их построения и оценки.

Если интерполяция кубическими сплайнами окажется неудовлетворительной, пользователю следует применить подпрограмму BSINT, позволяющую выбирать узлы интерполяции и задавать порядок интерполяционного сплайна.

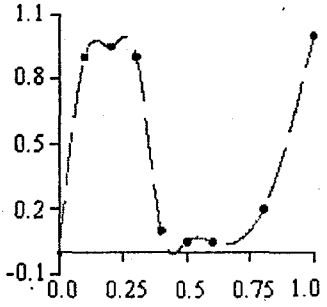
На рис. 1.1 приведены различные интерполяционные, в том числе и кубические, сплайны, построенные по следующим данным:

```
integer(4), parameter :: ndata = 9        ! Число точек
! Абсциссы точек
real(4) :: xdata(ndata) = (/ 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8, 1.0 /)
! Ординаты точек
real(4) :: fdata(ndata) = (/ 0.0, 0.9, 0.95, 0.9, 0.1, 0.05, 0.05, 0.2, 1.0 /)
```

В скобках надписи, поясняющей рисунок, указаны имена вычисляющих сплайны процедур.



а



б

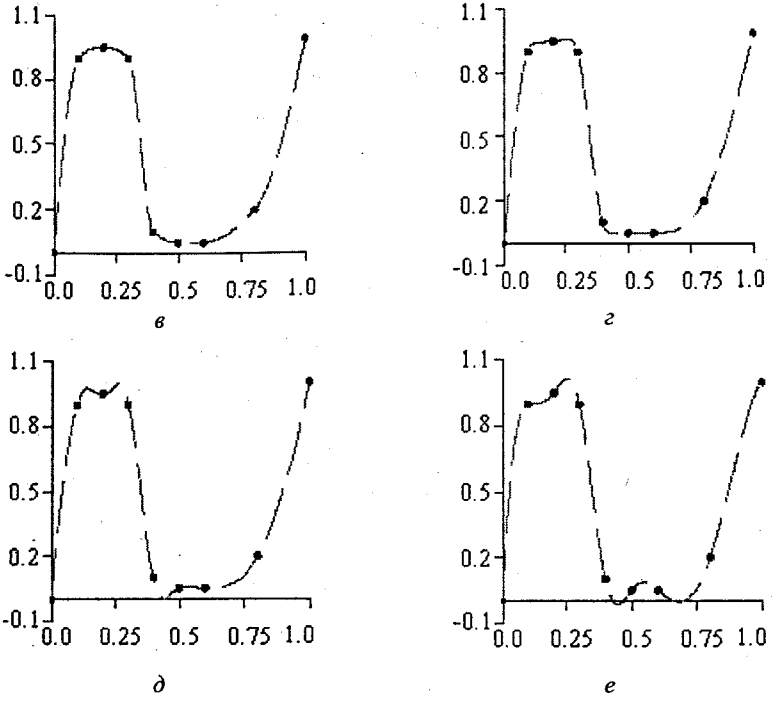


Рис. 1.1. Интерполяционные сплайны, построенные на одном наборе данных: а - интерполяционный кубический сплайн с граничными условиями "нет узла" (CSINT); б - естественный кубический сплайн (CSDEC); в - сплайн Акимы (CSAKM); г - интерполяционный кубический сплайн, сохраняющий выпуклость (CSCON); д - интерполяционный B-сплайн порядка k = 3 (BSINT); е - интерполяционный B-сплайн порядка k = 5 (BSINT)

Замечание. Только подпрограммы CSAKM и CSCON пытаются сохранить форму входной кривой. Остальные подпрограммы добавляют избыточные изгибы; максимальные колебания наблюдаются у B-сплайна, порядок которого $k = 5$.

1.2.4. ПРОСТРАНСТВЕННЫЕ СПЛАЙНЫ КАК ТЕНЗОРНОЕ ПРОИЗВЕДЕНИЕ

Простейший способ получения интерполяционного или сглаживающего сплайна, зависящего от нескольких переменных, состоит в применении некоторого метода, используемого для одной переменной, и в последующем получении результата как тензорного произведения зависящих от одной

переменной сомножителей. Результирующий сплайн называется *B-сплайн-тензорным произведением* или *ТП-B-сплайном.*

Рассмотрим двумерную интерполяцию. Пусть t_x - последовательность узлов для сплайнов порядка k_x, построенная на наборе точек $\{x_i\}_{i=1}^{n_x}$, а t_y - последовательность узлов для сплайнов порядка k_y, построенная на наборе точек $\{y_j\}_{j=1}^{n_y}$. (Размеры векторов t_x и t_y соответственно равны $n_x + k_x$ и $n_y + k_y$.) Тогда ТП-B-сплайн имеет вид:

$$\sum_{m=1}^{n_y}\sum_{n=1}^{n_x} b_{nm} B_{nk_x t_x}(x) B_{mk_y t_y}(y). \qquad (1.1)$$

Пусть на наборах точек $\{x_i\}_{i=1}^{n_x}$ и $\{y_j\}_{j=1}^{n_y}$ может быть решена задача вычисления коэффициентов одномерных B-сплайнов. Тогда задача интерполяции в виде ТП-B-сплайна формулируется следующим образом: необходимо найти коэффициенты b_{nm}, такие, что

$$\sum_{m=1}^{n_y}\sum_{n=1}^{n_x} b_{nm} B_{nk_x t_x}(x) B_{mk_y t_y}(y) = f_{ij}. \qquad (1.2)$$

Эта задача эффективно решается путем последовательного решения одномерных интерполяционных задач; детали см. в [7].

Аналогичным образом формулируется и решается задача для трехмерного случая (сплайн зависит от трех переменных).

1.2.5. КВАДРАТИЧНАЯ ИНТЕРПОЛЯЦИЯ

Подпрограммы, выполняющие квадратичную интерполяцию, т. е. интерполяцию в виде кусочно-параболической функции, начинаются с букв QD. Как и для кубических функций, процедуры квадратичной интерполяции находят сплайны для одномерного, двумерного и трехмерного случаев. Двумерные и трехмерные квадратичные сплайны вычисляются как тензорное произведение соответствующих одномерных сплайнов.

1.2.6. ИНТЕРПОЛЯЦИЯ ПО РАССЕЯННЫМ ДАННЫХ

Выполняется подпрограммой SURF, возвращающей коэффициенты интерполяционной поверхности пятой степени. Поверхность строится на треугольной сетке, является кусочно-многочленной и обладает C^1-непрерывностью. Процедура основана на работе [12].

1.2.7. МЕТОД НАИМЕНЬШИХ КВАДРАТОВ

Процедуры, формирующие сплайны по методу наименьших квадратов, ориентированы на работу с гладкими зашумленными данными. Эти процедуры находят:

- регрессию, т. е. зависимость среднего значения одной величины от некоторой другой величины или нескольких величин, используя многочлены произвольной степени (процедуры RLINE, RCURV) или пользовательскую функцию (процедура FNLSQ);
- B-сплайн по фиксированным (BSLSQ) или свободным (BSVLS) узлам;
- B-сплайн с линейными ограничениями (CONFT);
- ТП-B-сплайновые регрессии (BSLS2 и BSLS3).

1.2.8. СГЛАЖИВАНИЕ КУБИЧЕСКИМИ СПЛАЙНАМИ

В библиотеке есть две сглаживающие процедуры - CSSMH и CSSCV. Первая возвращает кубический сплайн, используя предоставленный пользователем сглаживающий параметр. Вторая самостоятельно оценивает сглаживающий параметр, а затем вычисляет сглаживающий кубический сплайн. В этом плане подпрограмма CSSCV более проста в употреблении. Также имеется подпрограмма CSSED, возвращающая сглаженную векторную аппроксимацию данных, засоренных случайными пиковыми выбросами.

1.2.9. РАЦИОНАЛЬНОЕ ЧЕБЫШЕВСКОЕ ПРИБЛИЖЕНИЕ

Выполняется подпрограммой RATCH с помощью заданной пользователем функцией. Подпрограмма может быть использована для поиска наилучшего приближения полиномами.

1.2.10. ПРИМЕНЕНИЕ ОДНОМЕРНЫХ ПРОЦЕДУР ДЛЯ СПЛАЙНОВ

Наиболее проста в употреблении подпрограмма CSIEZ, возвращающая интерполяционный кубический сплайн на заданной пользователем сетке. Подпрограмма SPLEZ дает возможность экспериментировать с различными интерполяционными и сглаживающими процедурами библиотеки IMSL.

В общем случае при решении задачи интерполяции (приближения) после построения соответствующего сплайна выполняется его оценка, дифференцирование или интегрирование. Выполнить эти задачи можно, применяя, например, подпрограмму CSINT, возвращающую коэффициенты кусочно-кубического сплайна с граничными условиями типа "нет узла", а затем выполняя оценку сплайна подпрограммой CSVAL.

Несколько сложнее вычислить интерполяционный B-сплайн, а затем многократно оценить его. Обычно задача решается в следующем порядке:

1) подпрограмма BSNAK возвращает узлы, необходимые для построения сплайна;
2) BSINT возвращает коэффициенты B-сплайна;
3) BSCPP преобразовывает его в КМ-представление;
4) PPVAL выполняет оценку КМ-представления B-сплайна.

Две последние подпрограммы (BSCPP и PPVAL) можно заменить оценщиком на сетке BS1GD.

Взаимодействие имеющихся в библиотеке IMSL 77 процедур для сплайнов отображено на рис. 1.2.

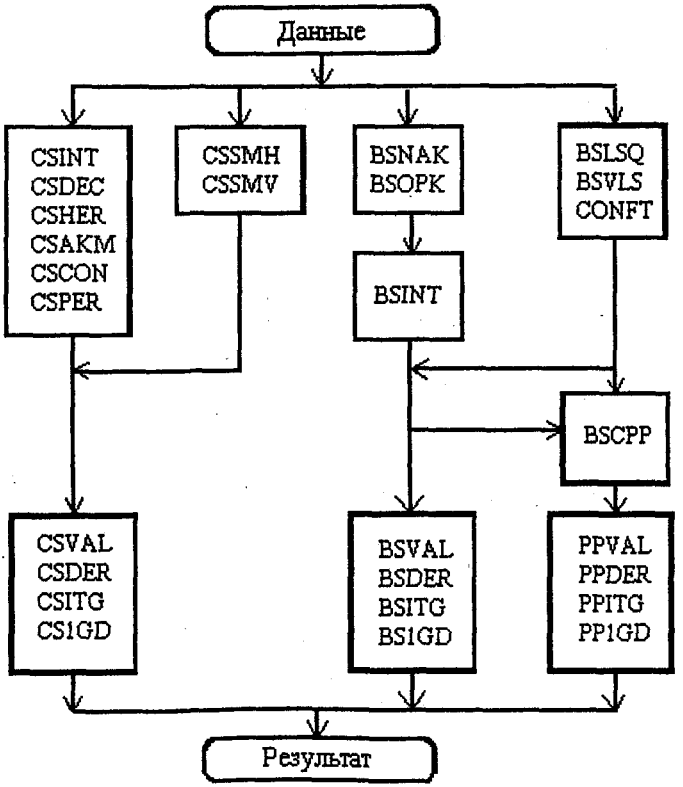


Рис. 1.2. Взаимосвязь процедур для сплайнов

1.2.11. ВЫБОР ИНТЕРПОЛЯЦИОННОЙ ПРОЦЕДУРЫ

Выбор выполняется из 18 интерполяционных процедур. Они отображены на рис. 1.3, который можно использовать и как руководство по выбору процедуры.

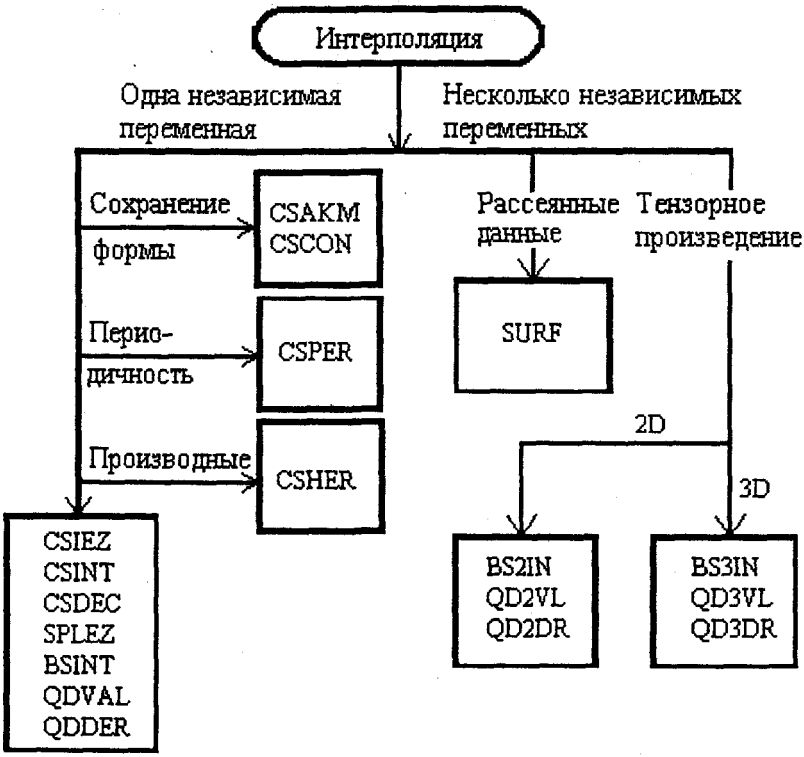


Рис. 1.3. Выбор интерполяционной процедуры

Например, в случае одномерных (с одной независимой переменной) периодических данных рис. 1.3 приводит к процедуре CSPER. Одномерная интерполяция без каких-либо особенностей выполняется одной из процедур нижнего левого прямоугольника рис. 1.3, например подпрограммой CSINT. И так далее.

1.3. ИНТЕРПОЛЯЦИЯ КУБИЧЕСКИМИ СПЛАЙНАМИ

1.3.1. ПЕРЕЧЕНЬ И ПАРАМЕТРЫ ПРОЦЕДУР

Вычисление и оценка интерполяционных кубических сплайнов осуществляется приведенными в табл. 1.1 процедурами библиотеки IMSL.

Таблица 1.1. Процедуры для интерполяционных кубических сплайнов

Процедура	*Описание*
	Вычисление сплайнов
CSINT	Находит интерполяционный кубический сплайн с граничными условиями "нет узла"
CSIEZ	Более удобная для использования подпрограмма, заменяющая связку CSINT плюс CSVAL. Находит интерполяционный кубический сплайн на заданной сетке с граничными условиями "нет узла"
CSDEC	Позволяет пользователю задавать различные условия в конечных точках, такие, как первая или вторая производная. Это означает, что естественный кубический сплайн можно получить, используя CSIEZ и CSDEC и задавая нулевую вторую производную на обоих концах отрезка интерполяции
CSHER	Находит интерполяционный кубический сплайн Эрмита. Должны быть известны и значения функции, и ее производные
CSAKM	Находит сплайн Акимы, форма которого наиболее приближена к форме кривой, задаваемой данными, стремясь при этом снизить колебания
CSCON	Находит интерполяционный кубический сплайн, форма которого наиболее приближена к форме кривой, обеспечивая выпуклость (вогнутость), присущую кривой, задаваемой входными данными
CSPER	Находит по периодическим данным периодический интерполяционный сплайн
	Оперирование сплайнами
CSVAL	Выполняет оценку кубического сплайна
CSDER	Возвращает значение производной кубического сплайна в заданной точке
CS1GD	Оценивает производную кубического сплайна на сетке
CSITG	Возвращает интеграл кубического сплайна

Поскольку многие процедуры для интерполяционных кубических сплайнов используют одинаковые имена параметров, то их целесообразно перечислить в одном месте - в табл. 1.2. Имена приводятся в алфавитном

порядке. Неповторяющиеся параметры и параметры, описание которых несколько отличается от описания одноименных параметров таблицы, даются при рассмотрении процедур.

Таблица 1.2. Параметры процедур для интерполяционных кубических сплайнов

Имя	*Смысл*	*Тип*
break	Вектор размера *ndata*, содержащий точки разрыва кубического сплайна. Элементы в *break* упорядочены по возрастанию значений. В процедурах, выполняющих оценку и интегрирование сплайнов, размер вектора *break* равен *nintv* + 1	REAL(4) или REAL(8)
cscoef	Массив формы (4, *ndata*), содержащий коэффициенты кусков кубического сплайна. В процедурах, выполняющих оценку и интегрирование сплайнов, размер массива *cscoef* имеет форму (4, *nintv* + 1)	То же
fdata	Вектор размера *ndata*, содержащий ординаты точек, по которым строится сплайн	"
ideriv	Порядок оцениваемой производной. Если *ideriv* = 0, то возвращается значение сплайна	INTEGER(4)
n	Размер вектора *xvec*	"
ndata	Число точек; не может быть менее двух	"
nintv	Число кусков, образующих сплайн	"
value	Вектор размера *n*, содержащий значения сплайна или его производной в точках вектора *xvec* (см., например, подпрограмму CS1GD)	REAL(4) или REAL(8)
x	Точка, в которой оценивается сплайн	То же
xdata	Вектор размера *ndata*, содержащий абсциссы точек. Абсциссы не должны повторяться	"
xvec	Вектор размера *n*, содержащий точки, в которых выполняется оценка сплайна. Данные в *xvec* располагаются в возрастающем порядке	"

1.3.2. ПОДПРОГРАММЫ, ВЫЧИСЛЯЮЩИЕ СПЛАЙНЫ

1.3.2.1. Подпрограмма CSIEZ (DCSIEZ)

Вычисляет кубический интерполяционный сплайн с граничными условиями "нет узла" и возвращает значения сплайна в заданных точках. Имеет вызов

CALL CSIEZ(*ndata, xdata, fdata, n, xvec, value*)

Параметр *value* является *выходным*. Остальные параметры - *входные*. Смысл параметров см. в табл. 1.2.

Описание:

Подпрограмма строит сплайн, возвращая не его коэффициенты, а значения сплайна в заданных точках. Дополнительные сведения о применяемых алгоритмах см. в разд. 1.3.2.2.

Пример. Вычисляется кубический интерполяционный сплайн по данным, порожденным функцией $f(x)$ = sin(15.0 * x) на отрезке [0, 1]. Значения сплайна сравниваются со значениями функции. Для графического представления сплайна используется режим векторного графа отображателя массивов (OM) фирмы Compaq.

```
program csiezTest
 use dfimsl
 integer(4), parameter :: ndata = 11
 integer(4) :: i
 real(4) :: f, fdata(ndata), value(2 * ndata - 1), x, xdata(ndata), xvec(2 * ndata - 1)
 ! Массив xyvalues введен для вывода графика сплайна
 real(kind(1e0)), allocatable :: xyvalues(:, :)
 !dec$attributes array_visualizer :: xyvalues
 f(x) = sin(15.0 * x)                    ! Задаем функцию
 do i = 1, ndata                         ! Формируем сетку
  xdata(i) = float(i - 1) / float(ndata - 1)
  fdata(i) = f(xdata(i))
 end do
 do i = 1, 2 * ndata - 1
  xvec(i) = float(i - 1) / float(2 * ndata - 2)
 end do
 ! Находим интерполяционный кубический сплайн и возвращаем его оценку
 call csiez(ndata, xdata, fdata, 2 * ndata - 1, xvec, value)
 ! Вывод заголовка
 write(*, "(13x, 'x', 9x, 'Interpolant', 5x, 'Error')")
 ! Вывод оценки сплайна и ошибки
 ! (выполняется с целью экономии места для x ∈ [0, 0.5])
 write(*, "(' ', 2f15.3, f15.6)") (xvec(i), value(i), f(xvec(i)) - value(i), i = 1, ndata)
 ! Вывод графика сплайна
 ! Используем режим векторного графа OM
 allocate(xyvalues(2, 2 * ndata - 1))
 xyvalues(1, :) = xvec
 xyvalues(2, :) = value
 call vGraph(xyvalues, 2 * ndata - 1)   ! Результат см. на рис. 1.4 при ndata = 45
 deallocate(xyvalues)
end program csiezTest
```

```
subroutine vGraph(fun, nvalues)        ! Выводит массив как векторный граф ОМ
 use avdef
 use avviewer
 use dflib
 integer(4) :: nvalues
 real(4) :: fun(2, nvalues)
 integer(4) hv, status, nError
 character(1) :: key
 character(av_max_label_len) :: xLabel = 'x'
 call faglStartWatch(fun, status)      ! Сообщаем ОМ имя отображаемого массива
 print *, "Starting Array Viewer"
 ! Запускаем ОМ с использованием fav-подпрограммы
 call favStartViewer(hv, status)
 if(status /= 0) then
  call favGetErrorNo(hv, nError, status)
  if(nError /= 0) then
   print *, "Array Viewer reports error ", nError
   stop
  end if
 end if
 ! Передаем ОМ данные подлежащего отображению массива
 call favSetArray(hv, fun, status)
 ! Задаем заголовок экземпляра ОМ
 call favSetArrayName(hv, "Spline 1", status)
 ! Отображаем массив в виде векторного графа
 call favSetGraphType(hv, VectorGraph, status)
 ! Задаем режим вывода заданных пользователем имен осей координат
 call favSetUseAxisLabel(hv, x_axis, 1, status)
 ! Новое (вместо dim1) имя x-оси координат. Длина переменной,
 ! задающей имя оси, равна AV_MAX_LABEL_LEN
 call favSetAxisLabel(hv, x_axis, xLabel, status)
 ! Показываем ОМ на экране
 call favShowWindow(hv, av_true, status)
 print *, "Press any key to close down the viewer"
 key = getcharqq( )
 call favEndViewer(hv, status)         ! Закрываем ОМ
 call faglEndWatch(fun, status)        ! Освобождаем ресурсы
end subroutine vGraph
```

Результат:

x	Interpolant	Error
0.000	0.000	0.000000
0.050	0.809	-0.127025

0.100	0.997	0.000000
0.150	0.723	0.055214
0.200	0.141	0.000000
0.250	-0.549	-0.022789
0.300	-0.978	0.000000
0.350	-0.843	-0.016246
0.400	-0.279	0.000000
0.450	0.441	0.009348
0.500	0.938	0.000000

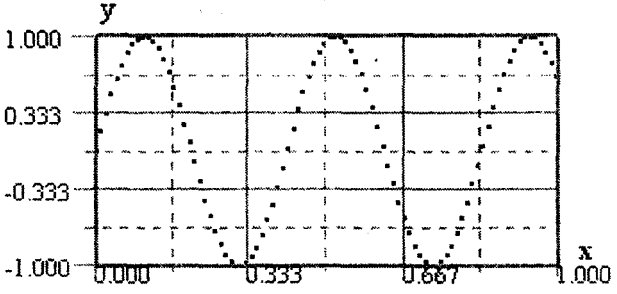


*Рис. 1.4. Кубический интерполяционный сплайн для функции f(x) = sin(15.0 * x)*

1.3.2.2. Подпрограмма CSINT (DCSINT)

Вычисляет кубический интерполяционный сплайн с граничными условиями "нет узла". Имеет вызов

CALL CSINT(*ndata, xdata, fdata, break, cscoef*)

Параметры подпрограммы CSINT:

Входные: ndata, xdata, fdata.

Выходные: break, cscoef.

Смысл параметров см. в табл. 1.2.

Комментарии:

1. Кубический сплайн можно оценить подпрограммой CSVAL, а его производные - подпрограммой CSDER.
2. Столбец *ndata* массива *cscoef* используется как рабочая память.

Приведенные комментарии справедливы для всех последующих подпрограмм, вычисляющих кубические интерполяционные сплайны.

Описание:

Подпрограмма CSINT вычисляет C^2-непрерывный кубический интерполяционный сплайн по набору точек (x_i, f_i) для $i = 1, \ldots, ndata$. (Далее вместо *ndata* будем использовать имя n.) Граничные условия типа "нет узла" определяются подпрограммой автоматически. Подобные конечные условия требуют непрерывности третьей производной сплайна во второй и предпоследней точках разрыва. Если $n = 2$ или $n = 3$, то в первом случае вычисляется линейная, а во втором - квадратичная интерполяция.

Если входные точки происходят от гладкой (скажем, C^4) функции f, то ошибка ведет себя предсказуемым образом. Пусть x - вектор точек разрыва. Тогда максимальная абсолютная ошибка удовлетворяет неравенству

$$\left\| f - S \right\|_{[\xi_1, \xi_n]} \le C \left\| f^{(4)} \right\|_{[\xi_1, \xi_n]} |\xi|^4, \qquad (1.3)$$

где

$$|\xi| = \max_{i=2,\ldots,n} |\xi_i - \xi_{i-1}|.$$

С деталями можно познакомиться в [7].

Пример. Вычисляется кубический интерполяционный сплайн для функции $f(x) = \sin(15x)$. Значения сплайна сравниваются со значениями функции.

```
program csintTest
 use dfimsl
 integer(4), parameter :: ndata = 11
 integer(4) :: i, nintv
 real(4) :: break(ndata), cscoef(4, ndata), f, fdata(ndata), x, xdata(ndata)
 f(x) = sin(15.0 * x)                    ! Задаем функцию
 do i = 1, ndata                         ! Формируем сетку
  xdata(i) = float(i - 1) / float(ndata - 1)
  fdata(i) = f(xdata(i))
 end do
 ! Находим интерполяционный кубический сплайн
 call csint(ndata, xdata, fdata, break, cscoef)
 ! Вывод заголовка таблицы результатов
 write(*, "(13x, 'x', 9x, 'Interpolant', 5x, 'Error')")
 nintv = ndata - 1
 do i = 1, 2 * ndata - 1                 ! Вывод оценки сплайна и ошибки
  x = float(i - 1) / float(2 * ndata - 2)
  write(*, "(2f15.3, f15.6)") x, csval(x, nintv, break, cscoef), f(x) - csval(x, nintv, break, cscoef)
 end do
end program csintTest
```

Результат тот же, что и для приведенной выше подпрограммы CSIEZ.

1.3.2.3. Подпрограмма CSDEC (DCSDEC)

Вычисляет кубический интерполяционный сплайн с заданными производными в конечных точках. Имеет вызов

CALL CSDEC(*ndata, xdata, fdata, ileft, dleft, iright, dright, break, cscoef*)

Параметры подпрограммы CSDEC:

Входные: ndata, xdata, fdata, ileft, dleft, iright, dright.

Выходные: break, cscoef.

ileft (*iright*) - тип граничного условия в левой (правой) конечной точке. Принимает следующие значения:

- 0 - условие "нет узла";
- 1 - параметр *dleft* (*dright*) задает первую производную;
- 2 - параметр *dleft* (*dright*) задает вторую производную.

dleft (*dright*) - значение производной (первой или второй) в левой (правой) граничной точке. Если *ileft* (*iright*) = 0, то *dleft* (*dright*) игнорируется.

Смысл остальных параметров подпрограммы CSDEC см. в табл. 1.2.

Комментарии см. в разд. 1.3.2.2.

Описание:

Справедливо описание, приведенное выше для подпрограммы CSINT. Дополнительно отметим, что условия в конечных точках выбираются и задаются пользователем. В каждой конечной точке пользователь задает одно из трех вышеупомянутых граничных условий *ileft* (*iright*).

С деталями можно познакомиться в [7, гл. 4 и 5].

Пример 1. Вычисляется кубический интерполяционный сплайн для функции $f(x) = \sin(15.0 * x)$. В левой конечной точке задается первая, а в правой - вторая производные. Значения сплайна сравниваются со значениями функции.

```
program csdecTest1
 use dfimsl
 integer(4), parameter :: ileft = 1, iright = 2, ndata = 11
 integer(4) :: i, nintv
 real(4) :: break(ndata), cscoef(4, ndata), dleft, dright, f, fdata(ndata), x, xdata(ndata)
 f(x) = sin(15.0 * x)                    ! Задаем функцию
 dleft = 15.0 * cos(15.0 * 0.0)          ! Задаем производные
 dright = -15.0 * 15.0 * sin(15.0 * 1.0)
 do i = 1, ndata                         ! Формируем сетку
  xdata(i) = float(i - 1) / float(ndata - 1)
```

```
 fdata(i) = f(xdata(i))
 end do
 ! Находим интерполяционный кубический сплайн
 call csdec(ndata, xdata, fdata, ileft, dleft, iright, dright, break, cscoef)
 ! Вывод заголовка
 write(*, "(13x, 'x', 9x, 'Interpolant', 5x, 'Error')")
 nintv = ndata - 1
 ! Вывод оценки сплайна и ошибки
 ! (выполняется для экономии места для x ∈ [0, 0.5])
 do i = 1, ndata
  x = float(i - 1) / float(2 * ndata - 2)
  write(*,"(2f15.3, f15.6)") x, csval(x, nintv, break, cscoef), f(x) - csval(x, nintv, break, cscoef)
 end do
end program csdecTest1
```

Результат:

x	Interpolant	Error
0.000	0.000	0.000000
0.050	0.675	0.006332
0.100	0.997	0.000000
0.150	0.759	0.019485
0.200	0.141	0.000000
0.250	-0.558	-0.013227
0.300	-0.978	0.000000
0.350	-0.840	-0.018765
0.400	-0.279	0.000000
0.450	0.440	0.009859
0.500	0.938	0.000000

Пример 2. Вычисляется естественный кубический интерполяционный сплайн для той же функции, что и в примере 1 настоящего раздела. Такой сплайн возвращается, если его вторая производная равна нулю на обоих концах. Как и ранее, значения сплайна сравниваются со значениями функции.

```
program csdecTest2
 use dfimsl
 integer(4), parameter :: ileft = 2, iright = 2, ndata = 11
 ! См. выше программу csdecTest1
 . . .
 dleft = 0.0; dright = 0.0                    ! Задаем вторые производные
 ! См. далее вышеприведенную программу csdecTest1
 . . .
end program csdecTest2
```

Результат:

x	Interpolant	Error
0.050	0.667	0.015027
0.100	0.997	0.000000
0.150	0.761	0.017156
0.200	0.141	0.000000
0.250	-0.559	-0.012609
0.300	-0.978	0.000000
0.350	-0.840	-0.018907
0.400	-0.279	0.000000
0.450	0.440	0.009812
0.500	0.938	0.000000

1.3.2.4. Подпрограмма CSHER (DCSHER)

Вычисляет кубический интерполяционный сплайн Эрмита. Имеет вызов

CALL CSHER(*ndata, xdata, fdata, dfdata, break, cscoef*)

Параметры подпрограммы CSHER:

Входные: ndata, xdata, fdata, dfdata.

Выходные: break, cscoef.

dfdata - вектор размера *ndata*, содержащий значения производной.

Смысл остальных параметров подпрограммы CSHER см. в табл. 1.2.

Комментарии:

1. При работе может возникнуть информационная ошибка типа 4 с кодом 2, означающая, что вектор *xdata* имеет повторяющиеся данные.
2. См. комментарии в разд. 1.3.2.2.

Описание:

Подпрограмма CSHER вычисляет C^1-непрерывный кубический интерполяционный сплайн по наборам данных (x_i, f_i) и (x_i, f_i') для $i = 1, \ldots, ndata$.

Если входные точки происходят от гладкой (скажем, C^4) функции f, то ошибка ведет себя предсказуемым образом и удовлетворяет неравенству (1.3).

С деталями можно познакомиться в [7].

Пример. Вычисляется кубический интерполяционный сплайн Эрмита для функции $f(x) = \sin(15x)$. Вместе со значениями функции подпрограмме передаются и значения ее производной. Значения сплайна сравниваются со значениями функции.

```
program csherTest
 use dfimsl
 integer(4), parameter :: ndata = 11
 integer(4) :: i, nintv
 real(4) :: break(ndata), cscoef(4, ndata), df, dfdata(ndata), f, fdata(ndata), x, xdata(ndata)
 f(x) = sin(15.0 * x)                    ! Задаем функцию и производную
 df(x) = 15.0 * cos(15.0 * x)
 do i = 1, ndata                         ! Формируем сетку
  xdata(i) = float(i - 1) / float(ndata - 1)
  fdata(i) = f(xdata(i))
  dfdata(i) = df(xdata(i))
 end do
 ! Находим интерполяционный кубический сплайн Эрмита
 call csher(ndata, xdata, fdata, dfdata, break, cscoef)
 ! Код для вывода результатов см. в примере для подпрограммы CSINT
 . . .
end program csherTest
```

Результат:

x	Interpolant	Error
0.000	0.000	0.000000
0.050	0.673	0.008654
0.100	0.997	0.000000
0.150	0.768	0.009879
0.200	0.141	0.000000
0.250	-0.564	-0.007257
0.300	-0.978	0.000000
0.350	-0.848	-0.010906
0.400	-0.279	0.000000
0.450	0.444	0.005714
0.500	0.938	0.000000

1.3.2.5. Подпрограмма CSAKM (DCSAKM)

Вычисляет кубический интерполяционный сплайн Акимы. Имеет вызов

CALL CSAKM(*ndata, xdata, fdata, break, cscoef*)

Параметры подпрограммы CSAKM:

Входные: ndata, xdata, fdata.

Выходные: break, cscoef.

Смысл параметров см. в табл. 1.2.

Комментарии см. в разд. 1.3.2.2.

Описание:

Подпрограмма CSAKM вычисляет C^1-непрерывный кубический интерполяционный сплайн по набору данных (x_i, f_i) для $i = 1, \ldots, ndata$. Граничные условия автоматически определяются подпрограммой (см. [7] или [12]).

Если входные точки происходят от гладкой (скажем, C^4) функции f, то ошибка ведет себя предсказуемым образом. Пусть x - вектор точек разрыва. Тогда максимальная абсолютная ошибка удовлетворяет неравенству

$$\|f - s\|_{[\xi_1, \xi_n]} \le C \left\| f^{(2)} \right\|_{[\xi_1, \xi_n]} |\xi|^2,$$

где $n = ndata$ и $|\xi| = \max_{i=2,\ldots,n} |\xi_i - \xi_{i-1}|$.

Подпрограмма CSAKM основана на методе Акимы [12], позволяющем бороться с избыточными изгибами сплайна (см. рис. 1.1 и замечание после рис. 1.1). Заметим, что, хотя сплайн является кусочно-кубическим, кубические многочлены им не воспроизводятся, но воспроизводятся линейные.

Приведем для оценки величины ошибки данные, которые получаются, если в примере для CSINT вызов подпрограммы CSINT заменить на вызов CSAKM.

x	I*nterpol*a*nt*	Error
0.000	0.000	0.000000
0.050	0.818	-0.135988
0.100	0.997	0.000000
0.150	0.615	0.163487
0.200	0.141	0.000000
0.250	-0.478	-0.093376
0.300	-0.978	0.000000
0.350	-0.812	-0.046447
0.400	-0.279	0.000000
0.450	0.386	0.064491
0.500	0.938	0.000000

1.3.2.6. Подпрограмма CSCON (DCSCON)

Находит интерполяционный кубический сплайн, форма которого наиболее приближена к форме кривой, обеспечивая выпуклость (вогнутость), присущую кривой, задаваемой входными данными. Имеет вызов

CALL CSCON(*ndata, xdata, fdata, ibreak, break, cscoef*)

Параметры подпрограммы CSCON:

Входные: ndata, xdata, fdata.

Выходные: ibreak, break, cscoef.

ibreak - число точек разрыва; будет меньше, чем 2**ndata*.

break - вектор размера 2**ndata*, содержащий в первых *ibreak* позициях точки разрыва кубического сплайна.

cscoef - массив формы (4, 2**ndata*), содержащий коэффициенты кусков кубического сплайна. Первые *ibreak* - 1 столбцов массива *cscoef* содержат локальные коэффициенты кубических составных частей кубического сплайна.

Смысл остальных параметров подпрограммы CSCON см. в табл. 1.2.

Комментарии:

1. При работе с CSCON могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	16	Превышено максимальное число итераций *itmax* (по умолчанию *itmax* = 25). Вызовите подпрограмму второго уровня C2CON (DC2CON), чтобы задать большее значение *itmax*
4	3	Вектор *xdata* имеет повторяющиеся данные

2. См. *комментарии* в разд. 1.3.2.2.

Описание:

Подпрограмма CSCON вычисляет интерполяционный кубический сплайн по $n = ndata$ точкам (x_i, f_i), $i = 1, \ldots, n$. Для простоты объяснения мы принимаем, что $x_i < x_{i+1}$, хотя пользователю совсем нет необходимости сортировать x_i. Если входные данные строго выпуклые, то таким же является и построенный подпрограммой CSCON сплайн. Кроме того, он C^2-непрерывен и минимизирует выражение

$$\int_{x_1}^{x_n} (g'')^2$$

по всем C^2-непрерывным функциям, интерполирующим данные. В общем случае, когда данные имеют и выпуклые и вогнутые области, сплайн также хорошо повторяет исходную кривую [36, 40]. Заметим, что, хотя сплайн является кусочно-кубическим, кубические многочлены им не воспроизводятся, но воспроизводятся линейные.

Одна из особенностей подпрограммы CSCON в том, что невозможно предсказать число точек разрыва в результирующем сплайне. В большинстве случаев точки разрыва не совпадают с точками в векторе *xdata*.

Пример. По одним и тем же данным подпрограммами CSCON и CSINT строятся сплайны. Поскольку при работе с подпрограммой CSCON превышается заданное по умолчанию максимальное число итераций, то CSCON выдает соответствующее сообщение.

```
program csconTest
 use dfimsl
 integer(4), parameter :: ndata = 9, nval = 50
 integer(4) :: i, ibreak                    ! Число точек разрыва
 real(4) :: break(2 * ndata), cscoef(4, 2 * ndata), fdata(ndata), xdata(ndata), x
 character(len = 2) :: clabel(14), rlabel(4)
 ! Массив xyvalues введен для вывода графика сплайна
 real(kind(1e0)), allocatable :: xyvalues(:, :)
 !dec$attributes array_visualizer :: xyvalues
 data xdata / 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8, 1.0 /
 data fdata / 0.0, 0.9, 0.95, 0.9, 0.1, 0.05, 0.05, 0.2, 1.0 /
 data rlabel / ' 1', ' 2', ' 3', ' 4' /
 data clabel / ' ', ' 1', ' 2', ' 3', ' 4', ' 5', ' 6', ' 7', ' 8', ' 9', '10', '11', '12', '13' /
 ! Находим интерполяционный сплайн
 call cscon(ndata, xdata, fdata, ibreak, break, cscoef)
 ! Выводим точки разрыва и массив коэффициентов cscoef
 write(*, '(1x, a, i2)') 'ibreak = ', ibreak
 call wrrrl('break', 1, ibreak, break, 1, 0, '(f8.2)', rlabel, clabel)
 call wrrrl('cscoef', 4, ibreak, cscoef, 4, 0, '(f8.2)', rlabel, clabel)
 ! Вывод графика сплайна, созданного подпрограммой CSCON
 ! Используем режим векторного графа ОМ
 allocate(xyvalues(2, 2 * nval - 1))
 ! Выполняем оценку сплайна в 2 * nval - 1 точках
 do i = 1, 2 * nval - 1
  x = float(i - 1) / float(2 * nval - 2)
  xyvalues(1, i) = x
  xyvalues(2, i) = csval(x, ibreak - 1, break, cscoef)
 end do
 call vGraph(xyvalues, 2 * nval - 1)    ! Результат см. на рис. 1.5, а
 deallocate(xyvalues)
 ! Находим интерполяционный кубический сплайн
 call csint(ndata, xdata, fdata, break, cscoef)
 ! Вывод графика сплайна, созданного подпрограммой CSINT
 allocate(xyvalues(2, 2 * nval - 1))
 ! Выполняем оценку сплайна в 2 * nval - 1 точках
 do i = 1, 2 * nval - 1
  x = float(i - 1) / float(2 * nval - 2)
  xyvalues(1, i) = x
```

```
  xyvalues(2, i) = csval(x, ndata - 1, break, cscoef)
 end do
 call vGraph(xyvalues, 2 * nval - 1)     ! Результат см. на рис. 1.5, б
 deallocate(xyvalues)
end program csconTest
```

Результат:

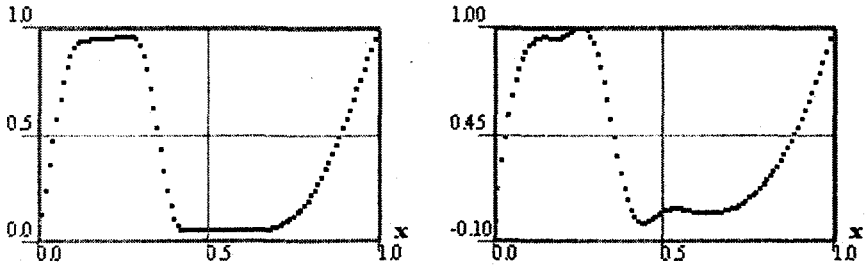
```
*** WARNING ERROR 16 from CSCON. The maximum number of iterations
***   was reached in Newton's Method. The best answer will be returned.
***   ITMAX = 25 was used, a larger value may help.
ibreak = 13
```

break

1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	0.1	0.14	0.2	0.26	0.3	0.4	0.44	0.5	0.6	0.61	0.8	1.0

cscoef

1	2	3	4	5	6	7	8	9	10	11	12	13
0.00	0.90	0.94	0.95	0.96	0.90	0.10	0.05	0.05	0.05	0.05	0.20	1.00
11.89	3.23	0.13	0.13	0.13	-4.43	-4.12	0.00	0.00	0.00	0.00	2.36	0.00
0.00	-173.17	0.00	0.00	0.00	220.22	226.47	0.00	0.00	0.00	0.00	24.66	0.00
-1731.74	841.60	0.00	0.00	-5312.08	4466.87	-6222.34	0.00	0.00	0.00	129.12	123.32	0.00



а *б*

Рис. 1.5. Сплайны, вычисленные разными подпрограммами:
а - создан подпрограммой CSCON; б - создан подпрограммой CSINT

1.3.2.7. Подпрограмма CSPER (DCSPER)

Находит интерполяционный кубический сплайн с периодическими граничными условиями. Имеет вызов

CALL CSPER(*ndata, xdata, fdata, break, cscoef*)

Параметры подпрограммы CSPER:

Входные: ndata, xdata, fdata.

Выходные: break, cscoef.

Смысл параметров подпрограммы CSPER см. в табл. 1.2.

Комментарии:

1. При работе может возникнуть информационная ошибка типа 3 с кодом 1, означающая, что вектор *xdata* содержит непериодические данные, т. е. первое и последнее значения *xdata* не равны. В этом случае используется первое значение вектора.
2. См. *комментарии* в разд. 1.3.2.2.

Описание:

Подпрограмма CSPER вычисляет C^2-непрерывный кубический интерполяционный сплайн, который имеет периодические граничные условия, т. е. для сплайна s справедливо $s(a) = s(b)$, $s'(a) = s'(b)$ и $s''(a) = s''(b)$, где a и b - соответственно начало и конец отрезка, на котором построен сплайн. Если ординаты в точках a и b не равны, то CSPER выдаст предупреждение и до начала вычислений ордината в точке b будет установлена равной ординате в точке a.

Если входные точки происходят от гладкой (скажем, C^4) функции f, то ошибка удовлетворяет неравенству (1.3).

С деталями можно ознакомиться в [7].

Пример. Вычисляется кубический интерполяционный сплайн для функции $f(x) = \sin(15x)$. Значения сплайна сравниваются со значениями функции.

```
program csperTest
 use dfimsl
 integer(4), parameter :: ndata = 11
 integer(4) :: i, nintv
 real(4) :: break(ndata), cscoef(4, ndata), csval, f, fdata(ndata), h, pi, x, xdata(ndata)
 f(x) = sin(15.0 * x)                    ! Задаем функцию
 pi = const('pi')
 h = 2.0 * pi / 15.0 / 10.0
 do i = 1, ndata                         ! Формируем сетку
   xdata(i) = h * float(i - 1)
   fdata(i) = f(xdata(i))
 end do
 ! Обеспечиваем периодические граничные условия
 fdata(ndata) = fdata(1)
 ! Находим интерполяционный кубический сплайн
 call csper(ndata, xdata, fdata, break, cscoef)
 ! Вывод заголовка
```

```
 write(*, "(13x, 'x', 9x, 'Interpolant', 5x, 'Error')")
 nintv = ndata - 1; h = h / 2.0
 do i = 1, 2 * ndata - 1                    ! Вывод оценки сплайна и ошибки
  x = h * float(i - 1)
  write(*, '(2f15.3, f15.6)') x, csval(x, nintv, break, cscoef),                &
       f(x) - csval(x, nintv, break, cscoef)
 end do
end program csperTest
```

Результат:

x	Interpolant	Error
0.000	0.000	0.000000
0.021	0.309	0.000138
0.042	0.588	0.000000
0.063	0.809	0.000362
0.084	0.951	0.000000
0.105	1.000	0.000447
0.126	0.951	0.000000
0.147	0.809	0.000362
0.168	0.588	0.000000
0.188	0.309	0.000138
0.209	0.000	0.000000

1.3.2.8. Подпрограмма SPLEZ (DSPLEZ)

Вычисляет интерполяционный или аппроксимирующий сплайн. Имеет вызов

CALL SPLEZ(*ndata, xdata, fdata, itype, ider, n, xvec, value*)

Параметры подпрограммы SPLEZ:

Параметр *value* является *выходным*. Остальные параметры - *входные*. Смысл параметров, кроме приведенных ниже, см. в табл. 1.2.

itype - задает тип сплайна; можно выбирать любое из допустимых значений *itype*, если *ndata* > 6. В зависимости от значения для вычисления сплайна в SPLEZ вызываются подпрограммы:

- CSINT, если *itype* = 1;
- CSAKM, если *itype* = 2;
- CSCON, если *itype* = 3;
- BSINT-BSNAK с *k* =2, если *itype* = 4;

- BSINT-BSNAK с k =3, если *itype* = 5;
- BSINT-BSNAK с k =4, если *itype* = 6;
- BSINT-BSNAK с k =5, если *itype* = 7;
- BSINT-BSNAK с k =6, если *itype* = 8;
- CSSCV, если *itype* = 9;
- BSLSQ с k = 2, если *itype* = 10;
- BSLSQ с k = 3, если *itype* = 11;
- BSLSQ с k = 4, если *itype* = 12;
- BSVLS с k = 2, если *itype* = 13;
- BSVLS с k = 3, если *itype* = 14;
- BSVLS с k = 4, если *itype* = 15.

ider - желаемый порядок производной.

Значение параметра *ndata* должно удовлетворять следующим условиям:

- *ndata* > *itype*, если *itype* = 1;
- *ndata* > 3, если *itype* = 2, 3;
- *ndata* > *itype* - 3, если *itype* = 4, 5, 6, 7, 8;
- *ndata* > 3, если *itype* = 9, 10, 11, 12;
- *ndata* > *korder*, если *itype* = 13, 14, 15.

Для параметра *value* справедливо: *value*(i) = s(*xvec*(i)), если *ider* = 0, *value*(i) = s'(*xvec*(i)), если *ider* = 1, и т. д., где s - построенный сплайн. То есть *value* содержит либо y-координаты сплайна, либо y-координаты его производной порядка *ider*.

Комментарий. При работе с SPLEZ могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
4	1	Вектор *xdata* имеет повторяющиеся данные
4	2	Вектор *xvec* имеет повторяющиеся данные

Описание:

Подпрограмма создана для того, чтобы пользователь имел возможность экспериментировать с различными процедурами интерполяции и сглаживания, употребляя одно имя SPLEZ. Подпрограмма SPLEZ вычисляет интерполяционный кубический сплайн по n = *ndata* точкам (x_i, f_i) для i = 1, …, n, если

itype = 1, …, 8. Если *itype* ≥ 9, вычисляется сглаживающий сплайн и, как вариант, сплайн, получаемый методом наименьших квадратов. Подпрограмма SPLEZ возвращает вектор *value*, содержащий *y*-координаты сплайна или его производной порядка *ider*. То есть

$$y_j = s^{(i)}(v_j), j = 1, \ldots, n,$$

где i = *ider*, v = *xvec*, y = *value*, а s - вычисленный сплайн.

Пример. Вычисляются в результате изменения параметра *itype* всевозможные сплайны и их первая производная для функции $f(x) = \sin(x^2)$. Для каждого сплайна и каждой его первой производной выводятся максимальные отклонения от истинных значений.

```
program splezTest
 use dfimsl
 integer(4), parameter :: ndata = 21, n = 2 * ndata - 1
 integer(4) :: i, ider, itype
 real(4) :: fdata(ndata), fpval(n), fvalue(n), value(n), xdata(ndata), xvec(n)
 real(4) :: emax1(15), emax2(15), f, fp
 f(x) = sin(x * x)                          ! Задаем функцию и ее производную
 fp(x) = 2 * x * cos(x * x)
 do i = 1, ndata                            ! Формируем сетку
  xdata(i) = 3.0 * (float(i - 1) / float(ndata - 1))
  fdata(i) = f(xdata(i))
 end do
 do i = 1, n
  xvec(i) = 3.0 * (float(i - 1) / float(2 * ndata - 2))
  fvalue(i) = f(xvec(i))
  fpval(i) = fp(xvec(i))
 end do
 write(*, "(4x, 'itype', 6x, 'Max error for f', 5x, 'Max error for f'", /)")
 ! Вызываем SPLEZ для каждого значения itype
 do itype = 1, 15
  do ider = 0, 1
   call splez(ndata, xdata, fdata, itype, ider, n, xvec, value)
   if(ider == 0) then                       ! Вычисляем максимальную ошибку
    value = -1.0 * fvalue + value
    emax1(itype) = abs(value(isamax(n, value, 1)))
   else
    value = -1.0 * fpval + value
    emax2(itype) = abs(value(isamax(n, value, 1)))
   end if
  end do
```

```
 write(*, '(i7, 2f20.6)') itype, emax1(itype), emax2(itype)
 end do
end program splezTest
```

Результат:

itype	Max error for f	Max error for f'
1	0.014082	0.658018
2	0.024682	0.897757
3	0.020896	0.813228
4	0.083615	2.168083
5	0.010403	0.508043
6	0.014082	0.658020
7	0.004756	0.228858
8	0.001070	0.077159
9	0.020896	0.813228
10	0.392603	6.047916
11	0.162793	1.983959
12	0.045404	1.582624
13	0.588370	7.680381
14	0.752475	9.673786
15	0.049340	1.713031

1.3.3. ОЦЕНКА И ИНТЕГРИРОВАНИЕ ИНТЕРПОЛЯЦИОННЫХ КУБИЧЕСКИХ СПЛАЙНОВ

1.3.3.1. Функция CSVAL (DCSVAL)

Возвращает значение кубического сплайна в точке *x*. Имеет вызов

result = CSVAL(*x, nintv, break, cscoef*)

Все параметры функции CSVAL являются *входными*. Смысл параметров функции CSVAL см. в табл. 1.2.

Результирующая переменная: CSVAL.

Описание:

Функция CSVAL оценивает кубический сплайн в заданной точке. Она является частным случаем процедуры PPDER, оценивающей сплайн произвольного порядка или его производную. Примеры употребления CSVAL см. выше.

1.3.3.2. Функция CSDER (DCSDER)

Возвращает значение производной кубического сплайна в заданной точке. Имеет вызов

result = CSDER(*ideriv, x, nintv, break, cscoef*)

Все параметры функции CSDER являются *входными*. Смысл параметров функции CSDER см. в табл. 1.2.

Результирующая переменная: CSDER.

Пример. Функция CSDER применяется для оценки первой и второй производных сплайна, вычисленного подпрограммой CSINT. Полученные значения сравниваются с точными значениями соответствующих производных.

```
program csderTest
 use dfimsl
 integer(4), parameter :: ndata = 10
 integer(4) :: i, nintv
 real(4) :: break(ndata), cddf, cdf, cf, cscoef(4, ndata), ddf, df, f, fdata(ndata), x, xdata(ndata)
 f(x) = sin(15.0*x)                          ! Задаем функцию и производные
 df(x) = 15.0 * cos(15.0 * x)
 ddf(x) = -225.0 * sin(15.0 * x)
 do i =1, ndata                              ! Формируем сетку
  xdata(i) = float(i - 1) / float(ndata - 1)
  fdata(i) = f(xdata(i))
 end do
 ! Вычисляем интерполяционный кубический сплайн
 call csint(ndata, xdata, fdata, break, cscoef)
 ! Вывод заголовка таблицы результатов
 write(*, "(9x, 'x', 8x, 's(x)', 5x, 'error', 6x, 's''(x)', 5x, 'error', 6x, 's''''(x)', 4x, 'error', /)")
 nintv = ndata - 1
 do i = 1, ndata                             ! Вывод части результата
  x = float(i - 1) / float(2 * ndata - 1)
  cf = csder(0, x, nintv, break, cscoef)
  cdf = csder(1, x, nintv, break, cscoef)
  cddf = csder(2, x, nintv, break, cscoef)
  write(*, "(f11.3, 3(f11.3, f11.6))") x, cf, f(x) - cf, cdf, df(x) - cdf, cddf, ddf(x) - cddf
 end do
end program csderTest
```

Результат:

x	s(x)	Error	s'(x)	Error	s''(x)	Error
0.000	0.000	0.000000	26.285	-11.28479	-379.458	379.457794
0.053	0.902	-0.192203	8.841	1.722460	-283.411	123.664734

0.105	1.019	-0.019333	-3.548	3.425718	-187.364	-37.628586
0.158	0.617	0.081009	-10.882	0.146207	-91.317	-65.824875
0.211	-0.037	0.021155	-13.160	-1.837700	4.730	-1.062027
0.263	-0.674	-0.046945	-10.033	-0.355268	117.916	44.391640
0.316	-0.985	-0.015060	-0.719	1.086203	235.999	-11.066727
0.368	-0.682	-0.004651	11.314	-0.409097	154.861	-0.365387
0.421	0.045	-0.011915	14.708	0.284042	-25.887	18.552732
0.474	0.708	0.024292	9.508	0.702690	-143.785	-21.041260

1.3.3.3. Подпрограмма CS1GD (DCS1GD)

Оценивает производную кубического сплайна на сетке. Имеет вызов

CALL CS1GD(*ideriv, n, xvec, nintv, break, cscoef, value*)

Параметр *value* является *выходным*. Остальные параметры - *входные*. Смысл параметров см. в табл. 1.2.

Комментарий. При работе может возникнуть информационная ошибка типа 4 с кодом 4, означающая, что данные в *xvec* расположены не по возрастанию их значений.

Описание:

Подпрограмма CS1GD функционирует так же, как и вызываемая в цикле функция CSDER, правда намного эффективнее. Подпрограмма CS1GD основана на процедуре PPVALU, приведенной в [7].

1.3.3.4. Функция CSITG (DCSITG)

Возвращает интеграл кубического сплайна. Имеет вызов

result = CSITG(*a, b, nintv, break, cscoef*)

Параметры функции CSITG:

Все параметры функции CS1GD являются *входными*.

Результирующая переменная: CSITG.

a, b - соответственно нижняя и верхняя границы интегрирования.

Смысл остальных параметров подпрограммы CSITG см. в табл. 1.2.

Пример. Первоначально подпрограммой CSINT вычисляется интерполяционный кубический сплайн для функции x^2, затем оцениваются интегралы сплайна на отрезках [0.0, 0.5] и [0.0, 2.0]. Поскольку CSINT использует граничное условие "нет узла", сплайн повторяет x^2. Следовательно, интегралы соответственно равны 1/24 и 8/3.

```
program csitgTest
 use dfimsl
 integer(4), parameter :: ndata = 10
 integer(4) :: i, nintv
 real(4) :: a, b, break(ndata), cscoef(4, ndata), error, exact,                    &
       f, fdata(ndata), fi, value, x, xdata(ndata)
 f(x) = x * x                              ! Задаем функцию и ее интеграл
 fi(x) = x * x * x / 3.0
 do i = 1, ndata                           ! Задаем сетку
  xdata(i) = float(i - 1) / float(ndata - 1)
  fdata(i) = f(xdata(i))
 end do
 ! Вычисляем интерполяционный кубический сплайн
 call csint(ndata, xdata, fdata, break, cscoef)
 a = 0.0; b = 0.5                          ! Вычисляем интеграл на отрезке [0.0, 0.5]
 nintv = ndata - 1
 value = csitg(a, b, nintv, break, cscoef)
 ! Точное значение интеграла и ошибка
 exact = fi(b) - fi(a)
 error = exact - value
 write(*, 1) a, b, value, exact, error     ! Вывод результата
 a = 0.0; b = 2.0                          ! Вычисляем интеграл на отрезке [0.0, 2.0]
 value = csitg(a, b, nintv, break, cscoef)
 ! Точное значение интеграла и ошибка
 exact = fi(b) - fi(a)
 error = exact - value
 write(*, 1) a, b, value, exact, error     ! Вывод результата
 1 format(' On the closed interval (', f3.1, ',', f3.1, ') we have:', /, 1x,       &
       'Computed integral = ', f10.5, /, 1x,                                      &
       'Exact integral = ', f10.5, /, 1x, 'Error ', ' = ', f10.6, /, /)
end program csitgTest
```

Результат:

```
On the closed interval (0.0, 0.5) we have:
Computed integral = 0.04167
Exact integral = 0.04167
Error = 0.000000
On the closed interval (0.0, 2.0) we have:
Computed integral = 2.66666
Exact integral = 2.66667
Error = 0.000007
```

1.4. ИНТЕРПОЛЯЦИЯ B-СПЛАЙНАМИ

1.4.1. ПЕРЕЧЕНЬ И ПАРАМЕТРЫ ПРОЦЕДУР

Процедуры, вычисляющие коэффициенты B-сплайнов, ТП-B-сплайнов для двумерных и трехмерных случаев, а также процедуры, предназначенные для оценки и преобразования B-сплайнов, приведены в табл. 1.3.

Таблица 1.3. Процедуры для B-сплайнов

Процедуры	*Описание*
Процедуры, возвращающие одномерные B-сплайны	
BSINT	Находит интерполяционный B-сплайн и возвращает его коэффициенты
BSNAK	Вычисляет сплайновую последовательность узлов с граничными условиями типа "нет узла"
BSOPK	Вычисляет оптимальную сплайновую последовательность узлов
Процедуры, оперирующие одномерными B-сплайнами	
BSVAL	Возвращает оценку B-сплайна в заданной точке x
BSDER	Возвращает производную B-сплайна в заданной точке x
BS1GD	Возвращает производную B-сплайна на сетке
BSITG	Возвращает интеграл B-сплайна
BSCPP	Преобразовывает одномерный сплайн из B-сплайнового представления в КМ-представление
Процедуры, возвращающие ТП-B-сплайны	
BS2IN и BS3IN	Возвращают коэффициенты ТП-B-сплайна, зависящего соответственно от двух и трех переменных
BSLS2 и BSLS3	Возвращают, используя метод наименьших квадратов, коэффициенты ТП-B-сплайна, зависящего соответственно от двух и трех переменных
Процедуры, оперирующие ТП-B-сплайнами	
BS2VL и BS3VL	Выполняют оценку двумерного (BS2VL) и трехмерного (BS3VL) ТП-B-сплайна
BS2DR и BS3DR	Осуществляют оценку частных производных различного порядка двумерного (BS2DR) и трехмерного (BS3DR) ТП-B-сплайна в заданной точке
BS2GD и BS3GD	То же, но в заданной сетке
BS2IG и BS3IG	Вычисляют интеграл двумерного (BS2IG) и трехмерного (BS3IG) ТП-B-сплайна

Параметры процедур для одномерных B-сплайнов приводятся в табл. 1.4, а для двумерных и трехмерных - в табл. 1.5. Имена упорядочены в алфавитном порядке.

Таблица 1.4. Параметры процедур для одномерных B-сплайнов

Имя	*Смысл*	*Тип*
a, b	Соответственно нижняя и верхняя границы интегрирования	REAL(4) или REAL(8)
break	Вектор, содержащий *nppcf* + 1 точек разрыва КМ-представления B-сплайна; размер вектора *break* не может быть меньше *ncoef* - *korder* + 2. В функциях оценки КМ-представления B-сплайна размер вектора *break* равен *nintv* + 1. Вектор *break* является строго возрастающим	То же
bscoef	Вектор размера *ndata*, содержащий коэффициенты B-сплайна. В процедурах, оценивающих B-сплайн и вычисляющих сплайн по методу наименьших квадратов, размер вектора *bscoef* равен *ncoef*	"
check	Логическая переменная; задается равной .TRUE., если необходима проверка вектора *xdata*, или .FALSE. - в противном случае	LOGICAL(4)
fdata	Вектор размера *ndata*, содержащий ординаты точек, по которым строится сплайн	REAL(4) или REAL(8)
ideriv	Порядок оцениваемой производной. Если *ideriv* = 0, то возвращается значение сплайна	INTEGER(4)
korder	Порядок B-сплайна; *korder* ≤ *ndata*	"
n	Размер вектора *xvec*	"
ncoef	Число коэффициентов B-сплайна; *ncoef* ≤ *ndata*	"
ndata	Число точек. Не может быть менее двух	"
nintv	Число кусков в КМ-представлении B-сплайна	"
nppcf	Число кусков в КМ-представлении сплайна; всегда *nppcf* ≤ *ncoef* - *korder* + 1	"
ppcoef	Массив формы (*korder, nppcf*), содержащий локальные коэффициенты многочленов из КМ-представления B-сплайна. В функциях оценки КМ-представления B-сплайна массив *ppcoef* имеет форму (*korder, nintv*)	REAL(4) или REAL(8)
value	Вектор размера *n*, содержащий значения сплайна или его производной в точках вектора *xvec* (см., например, подпрограмму BS1GD)	То же
x	Точка, в которой оценивается сплайн	"
xdata	Вектор размера *ndata*, содержащий абсциссы точек. Абсциссы не должны повторяться	"

xknot	Вектор размера *ndata* + *korder*, содержащий неубывающую последовательность узлов. В процедурах, оценивающих и преобразовывающих B-сплайн, размер вектора *xknot* равен *korder* + *ncoef*	REAL(4) или REAL(8)
xvec	Вектор размера *n*, содержащий точки, в которых выполняется оценка сплайна. Данные в *xvec* располагаются в возрастающем порядке	"

Таблица 1.5. Параметры процедур для двумерных и трехмерных B-сплайнов

Имя	*Смысл*	*Тип*
a, b	Соответственно нижняя и верхняя границы интегрирования по *x*	REAL(4) или REAL(8)
bscoef	Массив формы (*nxdata, nydata*), содержащий коэффициенты B-сплайна. В процедурах, оценивающих двумерный B-сплайн, *bscoef* имеет форму (*nxcoef, nycoef*). В случае трехмерного сплайна *bscoef* - это массив формы (*nxdata, nydata, nzdata*), содержащий коэффициенты B-сплайна. В процедурах, оценивающих трехмерный B-сплайн, *bscoef* имеет форму (*nxcoef, nycoef, nzcoef*)	То же
c, d	Соответственно нижняя и верхняя границы интегрирования по *y*	"
check	Логическая переменная; задается равной .TRUE., если необходима проверка вектора *xdata* и *ydata*, или .FALSE. - в противном случае	LOGICAL(4)
e, f	Соответственно нижняя и верхняя границы интегрирования по *z*	REAL(4) или REAL(8)
fdata	Массив формы (L*df, nydata*), содержащий *nxdata*×*nydata* значений функции. Причем *fdata*(*i, j*) есть значение в точке (*xdata*(*i*), *ydata*(*j*)). В случае трехмерного сплайна *fdata* - это массив формы (L*df, mdf, nzdata*), содержащий *nxdata*×*nydata*×*nzdata* значений функции. Причем *fdata*(*i, j, k*) есть значение в точке (*xdata*(*i*), *ydata*(*j*), *zdata*(*k*))	То же
ixder (*iyder, izder*)	Порядок производной в направлении *x* (*y, z*). Если *ixder* = *iyder* = *izder* = 0, то возвращается значение сплайна	INTEGER(4)
kxord (*kyord, kzord*)	Порядок сплайна в направлении *x* (*y*); *kxord* ≤ *nxdata*, *kyord* ≤ *nydata*	"
L*df*	Ведущий размер массива *fdata*	"
L*dvalu*	Ведущий размер массива *value*	

mdf	Средний размер массива *fdata* (случай трехмерного сплайна)	INTEGER(4)
mdvalu	Средний размер массива *value* (случай трехмерного сплайна)	
nxcoef (*nycoef, nzcoef*)	Протяженность массива *bscoef* по измерению *x* (*y, z*)	"
nx (*ny, nz*)	Размер вектора *xvec* (*yvec, zvec*) (число точек по *x* (*y, z*))	"
nxdata (*nydata, nzdata*)	Число точек по *x* (*y, z*)	"
value	Массив формы (*nx, ny*), содержащий значения (*ixder, iyder*)-частной производной сплайна. Причем *value*(*i, j*) содержит производную сплайна в точке (*xvec*(*i*), *yvec*(*j*)). В случае трехмерного сплайна *value* - это массив формы (*nx, ny, nz*), содержащий значения (*ixder, iyder, izder*)-частной производной сплайна. Причем *value*(*i, j, k*) содержит производную сплайна в точке (*xvec*(*i*), *yvec*(*j*), *zvec*(*k*))	REAL(4) или REAL(8)
(*x, y*)	Точка, в которой оценивается двумерный сплайн	То же
(*x, y, z*)	Точка, в которой оценивается трехмерный сплайн	"
xdata (*ydata, zdata*)	Вектор размера *nxdata* (*nydata, nzdata*), содержащий координаты точек по *x* (*y*). Данные в *xdata* (*ydata, zdata*) должны быть строго возрастающими	"
xknot (*yknot, zknot*)	Вектор размера *nxdata* + *kxord* (*nydata* + *kyord, nzdata* + *kzord*), содержащий последовательность узлов в направлении *x* (*y, z*). Значения в *xknot* (*yknot, zknot*) расположены в неубывающем порядке. В процедурах, оценивающих и вычисляющих B-сплайн по методу наименьших квадратов, размер вектора *xknot* (*yknot, zknot*) равен *nxcoef* + *kxord* (*nycoef* + *kyord, nzcoef* + *kzord*)	"
xvec (*yvec, zvec*)	Вектор размера *nx* (*ny, nz*), содержащий точки, в которых выполняется оценка сплайна. Данные в *xvec* (*yvec, zvec*) располагаются в возрастающем порядке	"

1.4.1. ОБОЗНАЧЕНИЯ В ФОРМУЛАХ

В формулах для B-сплайнов используются следующие обозначения:

b - массив *bscoef*;

$B_j = B_{jkt}$ - *j*-й B-сплайн порядка *k*, относящийся к последовательности узлов *t*;

$B_{jk_x t_x}$ ($B_{jk_y t_y}$, $B_{jk_z t_z}$) - *j*-й B-сплайн порядка k_x (k_y, k_z), относящийс к последовательности узлов t_x (t_y, t_z);

f - вектор ординат *fdata*;

k - порядок сплайна *korder*;

k_x (k_y, k_z) - порядок сплайна *kxord* (*kyord*, *kzord*);

n - число точек *ndata*;

n_c - число коэффициентов *ncoef* B-сплайна;

n_{cx} (n_{cy}, n_{cz}) - протяженность массива *bscoef* по измерению *x* (*y*, *z*);

n_x (n_y, n_z) - число точек *nxdata* (*nydata*, *nzdata*);

s - вычисленный сплайн;

t - последовательность узлов *xknot* (случай одномерного сплайна);

t_x (t_y, t_z) - последовательность узлов *xknot* (*yknot*, *zknot*);

x (*y*, *z*) - вектор *xdata* (*ydata*, *zdata*).

1.4.1. ПОДПРОГРАММЫ, ВЫЧИСЛЯЮЩИЕ B-СПЛАЙНЫ

1.4.1.1. Подпрограмма BSINT (DBSINT)

Находит интерполяционный B-сплайн и возвращает его коэффициенты. Имеет вызов

CALL BSINT(*ndata, xdata, fdata, korder, xknot, bscoef*)

Параметр *bscoef* является *выходным*. Остальные параметры - *входные*. Смысл параметров см. в табл. 1.4.

Комментарии:

1. При работе с BSINT могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	1	Интерполяционная матрица плохо обусловлена
4	3	Вектор *xdata* имеет повторяющиеся данные
4	4	Число повторяемости узлов не может превышать порядок сплайна
4	5	Узлы должны быть упорядочены по неубыванию значений
4	15	Наименьший *i*-й элемент вектора *xdata* должен быть больше *i*-го узла и меньше узла *i* + *korder*
4	16	Наибольший элемент вектора *xdata* должен быть больше узла *ndata* и меньше узла *ndata* + *korder* или равен ему
4	17	Наименьший элемент вектора *xdata* должен быть больше первого узла или равен ему и меньше узла *korder* + 1

2. B-сплайн можно оценить функцией BSVAL, а его производные BSDER.

Описание:

Подпрограмма BSINT вычисляет вектор b = *bscoef*, удовлетворяющий линейной системе

$$\sum_{j=1}^{n} b_j B_j(x_i) = f_i .$$

Приведенная линейная система является ленточной с не менее k - 1 нижних и k - 1 верхних кодиагоналей. Матрица

$$A = (B_j(x_j))$$

положительно определенная и имеет обратную матрицу, когда диагональные элементы отличны от нуля.

Подпрограмма BSINT основана на процедуре SPLINT, приведенной в [7]. Подпрограмма BSINT возвращает коэффициенты интерполяционного B-сплайна порядка *korder* с последовательностью узлов *xknot* по точкам (x_i, f_i) для $i = 1, \ldots ndata$.

Первоначально BSINT сортирует вектор *xdata* и заносит результат в x. Элементы вектора *fdata* перестанавливаются соответствующим образом и запоминаются в f. При этом выполняются следующие проверки:

$$x_i < x_{i+1}, \qquad i = 1, \ldots n - 1;$$

$$t_i < t_{i+1}, \qquad i = 1, \ldots n;$$

$$t_i \le t_{i+k}, \qquad i = 1, \ldots n + k - 1.$$

Чтобы убедиться в том, что интерполяционная матрица невырожденная, выполняется проверка $t_k \le x_i \le t_{n+1}$ для $i = 1, \ldots n$.

В общем случае неизвестно точное поведение ошибки интерполяции. Однако если t и x выбраны надлежащим образом и входные точки происходят от гладкой (скажем, C^k) функции f, то ошибка ведет себя предсказуемым образом. Максимальная абсолютная ошибка для сплайна s удовлетворяет неравенству

$$\left\| f - s \right\|_{[t_k, t_{n+1}]} \le C \left\| f^{(k)} \right\|_{[t_k, t_{n+1}]} \left| t \right|^k ,$$

где

$$\left| t \right| = \max_{i=k,\ldots,n} \left| t_{i+1} - t_i \right| .$$

С деталями можно познакомиться в [7, гл. 13].

Подпрограмма BSINT может быть использована вместо CSINT, если вызвать подпрограмму BSNAK, возвращающую надлежащие узлы, затем - подпрограмму BSINT для получения коэффициентов B-сплайна и в завершение - подпрограмму BSCPP, преобразовывающую B-сплайн в КМ-форму.

Пример. Вычисляется интерполяционный B-сплайн *s* для функции $f(x) = \sqrt{x}$. Процедурой BSVAL выполняется сравнение ординат сплайна с точными значениями функции.

```
program bsintTest
 use dfimsl
 integer(4), parameter :: korder = 3, ndata = 5, nknot = ndata + korder
 integer(4) :: i, ncoef
 real(4) :: bscoef(ndata), bt, f, fdata(ndata), x, xdata(ndata), xknot(nknot), xt
 f(x) = sqrt(x)                          ! Задаем функцию
 do i = 1, ndata                         ! Задаем точки интерполяции
  xdata(i) = float(i - 1) / float(ndata - 1)
  fdata(i) = f(xdata(i))
 end do
 ! Генерируем последовательность узлов
 call bsnak(ndata, xdata, korder, xknot)
 ! Выполняем интерполяцию
 call bsint(ndata, xdata, fdata, korder, xknot, bscoef)
 ! Вывод заголовка
 write(*, "(/, 6x, 'x', 19x, 's(x)', 18x, 'error', /)")
 ! Подготовка к оценке сплайна в точке xdata(1)
 ncoef = ndata
 xt = xdata(1)
 ! Оценка сплайна
 bt = bsval(xt, korder, xknot, ncoef, bscoef)
 write(*, 99998) xt, bt, f(xt) - bt
 do i = 2, ndata                         ! Оценка сплайна в последующих точках
  xt = (xdata(i - 1) + xdata(i)) / 2.0
  bt = bsval(xt, korder, xknot, ncoef, bscoef)
  write(*, 99998) xt, bt, f(xt) - bt
  xt = xdata(i)
  bt = bsval(xt, korder, xknot, ncoef, bscoef)
  write(*, 99998) xt, bt, f(xt) - bt
 end do
 99998 format(' ', f6.4, 15x, f8.4, 12x, f11.6)
end program bsinetTest
```

Результат:

Vector xknot:

0.00 0.00 0.00 0.38 0.62 1.00 1.00 1.00

x	*s*(x)	Error
0.0000	0.0000	0.000000
0.1250	0.2918	0.061781
0.2500	0.5000	0.000000
0.3750	0.6247	-0.012311
0.5000	0.7071	0.000000
0.6250	0.7886	0.002013
0.7500	0.8660	0.000000
0.8750	0.9365	-0.001092
1.0000	1.0000	0.000000

1.4.1.2. Подпрограмма BSNAK (DBSNAK)

Вычисляет сплайновую последовательность узлов с граничными условиями типа "нет узла". Имеет вызов

CALL BSNAK(*ndata, xdata, korder, xknot*)

Параметр *xknot* является *выходным*. Остальные параметры - *входные*. Смысл параметров см. в табл. 1.4.

Комментарии:

1. Может возникнуть информационная ошибка типа 4 с кодом 4, означающая, что вектор *xdata* имеет повторяющиеся данные.
2. Первый узел находится в левой конечной точке, последний - недалеко за правой конечной точкой. Каждый конечный узел повторяется *korder* раз. Внутренние узлы не дублируются.

Описание:

Вектор t = *xknot* в первых его $n + k$ (k = *korder*) позициях содержит последовательность узлов, позволяющую найти по входным точкам интерполяционный сплайн порядка k. Если k четно, то (предполагаем, что данные в векторе x = *xdata* расположены по возрастанию значений)

$$t_i = x_1 \text{ для } i = 1, \ldots, k;$$

$$t_i = x_{i-k/2} \text{ для } i = k+1, \ldots, n;$$

$$t_i = x_{n+\varepsilon} \text{ для } i = n+1, \ldots, n+k,$$

где ε - малая положительная константа. Если k нечетно, тогда

$t_i = x_1$ для $i = 1, \ldots, k$;

$t_i = (x_{i-(k-1)/2} + x_{i-1-(k-1)/2})/2$ для $i = k+1, \ldots, n$;

$t_i = x_{n+\varepsilon}$ для $i = n+1, \ldots, n+k$.

Заметим, что нет необходимости сортировать вектор *xdata*, поскольку все равно сортировка выполняется подпрограммой BSNAK.

Пример употребления BSNAK см. выше.

1.4.1.3. Подпрограмма BSOPK (DBSOPK)

Вычисляет оптимальную сплайновую последовательность узлов. Имеет вызов

CALL BSOPK(*ndata, xdata, korder, xknot*)

Описание параметров то же, что и для BSNAK.

Комментарии:

1. При работе с BSOPK могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	6	Ньютоновский итерационный метод не сходится
4	3	Вектор *xdata* имеет повторяющиеся данные
4	4	Интерполяционная матрица вырожденная. Возможно, данные *xdata* расположены слишком близко друг к другу

2. Заданное по умолчанию максимальное число итераций *maxit* = 10 ньютоновского метода можно изменить, вызвав непосредственно процедуру второго уровня B2OPK (DB2OPK).

Описание:

По данным абсциссам x и порядку сплайна k подпрограмма BSOPK возвращает последовательность узлов t, такую, что минимизируется константа c в формуле оценки ошибки

$$\|f - s\| \le c\|f^{(k)}\|,$$

где f - произвольная C^k-непрерывная функция, а s - интерполяционный сплайн для функции f, заданной в точках x с последовательностью узлов t. Алгоритм основывается на приведенных в [7] сведениях.

Пример. На отрезке [0, 1] для функции $f(x) = \sin(10x^3)$ вычисляются 6 интерполяционных B-сплайнов s_k ($k = 3, \ldots, 8$). Подпрограмма BSOPK генерирует узлы, а BSINT находит сплайны. Для каждого k на 100 равномерно размещенных точках оценивается абсолютная шибка

$$|s_k - f|,$$

максимальное значение которой выводится на печать. Также в таблице результатов приводятся ошибки, получаемые при использовании BSNAK взамен BSOPK.

```
program bsopkTest
 use dfimsl
 integer(4), parameter :: kmax = 8, kmin = 3, ndata = 20
 integer(4) :: i, k, korder, j
 real(4) :: bscoef(ndata), bsval, dif, difmax(kmax - kmin + 1), f,                &
       fdata(ndata), ft, st, t, x, xdata(ndata), xknot(kmax + ndata), xt
 f(x) = sin(10.0 * x * x * x)          ! Определяем функцию и тау-функцию,
 t(x) = 1.0 - x * x                    ! которую используем для получения xdata
 do i = 1, ndata                       ! Генерируем входные данные
  xt = float(i - 1) / float(ndata - 1)
  xdata(i) = t(xt)
  fdata(i) = f(xdata(i))
 end do
 difmax = 0.0
 ! Цикл, задающий различные порядки сплайнов
 do k = kmin, kmax
  korder = k
  j = k - kmin + 1
  ! Получаем узлы и выполняем интерполяцию
  call bsopk(ndata, xdata, korder, xknot)
  call bsint(ndata, xdata, fdata, korder, xknot, bscoef)
  do i =1, 100                         ! st - оценка сплайна
   xt = float(i - 1) / 99.0
   st = bsval(xt, korder, xknot, ndata, bscoef)
   ft = f(xt)
   dif = abs(ft - st)
   ! Максимальная разность
   difmax(j) = amax1(dif, difmax(j))
  end do                               ! Вывод результата
 end do
 write(*, "(a, 15x, 10i9)") ' korder', (k, k = kmin, kmax)
 write(*, "(a, 3x, 10f9.4)") ' Maximum difference', (difmax(j), j = 1, kmax - kmin + 1)
end program bsopkTest
```

Результат:

korder	3	4	5	6	7	8
Подпрограмма BSOPK						
Maximum difference	0.0096	0.0018	0.0005	0.0004	0.0007	0.0039
Подпрограмма BSNAK						
Maximum difference	0.0080	0.0026	0.0004	0.0008	0.0010	0.0004

1.4.1.4. Подпрограмма BS2IN (DBS2IN)

Вычисляет двумерный интерполяционный ТП-В-сплайн, возвращая его коэффициенты. Имеет вызов

CALL BS2IN(*nxdata, xdata, nydata, ydata, fdata,* &
Ldf, kxord, kyord, xknot, yknot, bscoef)

Параметр *bscoef* является *выходным*. Остальные параметры - *входные*. Смысл параметров см. в табл. 1.5.

Комментарий. При работе с BS2IN могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	1	Интерполяционная матрица почти вырожденная; *LU*-разложение выполнить не удается
3	2	Интерполяционная матрица почти вырожденная; итерационное уточнение выполнить не удается
4	6	Значения в *xdata* должны быть строго возрастающими
4	7	Значения в *ydata* должны быть строго возрастающими
4	13	Число повторяемости узлов не может превышать порядок сплайна
4	14	Узлы должны быть упорядочены по неубыванию их значений
4	15	Наименьший *i*-й элемент вектора *xdata* (*ydata*) должен быть больше *i*-го узла и меньше узла *i* + *kxord* (*kyord*)
4	16	Наибольший элемент вектора *xdata* (*ydata*) должен быть больше узла *ndata* и меньше узла *ndata* + *kxord* (*kyord*) или равен ему
4	17	Наименьший элемент вектора *xdata* (*ydata*) должен быть больше первого узла или равен ему и меньше узла *kxord* (*kyord*) + 1
4	18	Данные в *xdata* должны быть строго возрастающими
4	19	Данные в *ydata* должны быть строго возрастающими

Описание:

Подпрограмма BS2IN вычисляет интерполяционный ТП-В-сплайн в виде (1.1). Алгоритм требует, чтобы

$$t_x(k_x) \le x_i \le t_x(n_x + 1),\ 1 \le i \le n_x;$$

$$t_y(k_y) \le y_j \le t_y(n_y + 1),\ 1 \le j \le n_y.$$

Задача вычисления ТП-В-сплайна сводится к решению линейной системы (1.2). Подпрограмма BS2IN основана на процедуре SPLI2D, приведенной в [7].

Пример. Вычисляется интерполяционный ТП-В-сплайн для функции $f(x, y) = x^3 + x * y$. В качестве результата выводятся значения сплайна, ошибки на сетке 4×4 и график В-сплайна на сетке [0.0, 2.0]×[0.0, 1.0] в *nxdata*×*nydata* точках. Для вывода графика применяется векторный режим ОМ.

```
program bs2inTest
 use dfimsl
 integer(4), parameter :: kxord = 5, kyord = 2, nxdata = 21, nxvec = 4, nydata = 6,     &
      nyvec = 4, Ldf = nxdata, nxknot = nxdata + kxord, nyknot = nydata + kyord
 integer(4) :: i, j, nxcoef, nycoef, kb
 real(4) :: bscoef(nxdata, nydata), f, fdata(Ldf, nydata), value(nxvec, nyvec), x,     &
      xdata(nxdata), xknot(nxknot), xvec(nxvec), y, ydata(nydata),                     &
      yknot(nyknot), yvec(nyvec)
 ! Bspline - массив, введенный для графического отображения
 ! В-сплайна средствами ОМ
 real(4) :: Bspline(3, nxdata * nydata)
 f(x, y) = x * x * x + x * y                    ! Задаем функцию и входные данные по x
 do i = 1, nxdata
  xdata(i) = float(i - 11) / 10.0
 end do
 ! Генерируем последовательность узлов x
 call bsnak(nxdata, xdata, kxord, xknot)
 ! Генерируем входные данные и последовательность узлов y
 do i = 1, nydata
   ydata(i) = float(i - 1) / 5.0
 end do
 call bsnak(nydata, ydata, kyord, yknot)
 do i = 1, nydata                               ! Генерируем fdata
  do j = 1, nxdata
    fdata(j, i) = f(xdata(j), ydata(i))
  end do
 end do                                         ! Интерполяция
 call bs2in(nxdata, xdata, nydata, ydata, fdata, Ldf, kxord, kyord, xknot, yknot, bscoef)
 nxcoef = nxdata; nycoef = nydata               ! Вывод заголовка
 write(*, "(13x, 'x', 14x, 'y', 10x, 's(x, y)', 9x, 'Error')")
 ! Вывод в области [0.0, 1.0]×[0.0, 1.0] в 16 точках
 do i = 1, nxvec
   xvec(i) = float(i - 1) / 3.0
 end do
 do i = 1, nyvec
   yvec(i) = float(i - 1) / 3.0
 end do
 ! Оцениваем сплайн
```

```
 call bs2gd(0, 0, nxvec, xvec, nyvec, yvec, kxord, kyord, xknot, yknot,                &
      nxcoef, nycoef, bscoef, value, nxvec)
 do i = 1, nxvec
  do j = 1, nyvec
   write(*, "(3f15.4, f15.6)") xvec(i), yvec(j), value(i, j), (f(xvec(i), yvec(j)) - value(i, j))
  end do
 end do
 ! Вывод графика в области [0.0, 2.0]×[0.0, 1.0] в nxdata×nydata точках
 ! Оцениваем сплайн в nxdata×nydata точках
 call bs2gd(0, 0, nxdata, xdata, nydata, ydata, kxord, kyord, xknot, yknot,            &
      nxcoef, nycoef, bscoef, fdata, nxdata)
 ! Графическое отображение результата приведено на рис. 1.6
 kb = 1
 do i = 1, nxdata
  Bspline(1, kb:kb + nydata - 1) = xdata(i)
  Bspline(2, kb:kb + nydata - 1) = ydata(1:nydata)
  Bspline(3, kb:kb + nydata - 1) = fdata(i, 1:nydata)
  kb = kb + nydata
 end do                                  ! Текст подпрограммы vGraph2 см. ниже
 ! Вызов OM
 call vGraph2(Bspline, nxdata * nydata)
end program bs2inTest

subroutine vGraph2(fun, nvalues)         ! Выводит массив как векторный граф OM
 use avdef
 use avviewer
 use dflib
 integer(4) :: nvalues                   ! Подпрограмма vGraph2 выводит в режиме
 real(4) :: fun(3, nvalues)              ! векторного графа трехмерный образ,
 integer(4) hv, status, nError           ! в то время как подпрограмма vGraph
 character(1) :: key                     ! формирует плоское изображение
 character(av_max_label_len) :: xLabel = 'x', yLabel = 'y', zLabel = 'z'
 call faglStartWatch(fun, status)        ! Сообщаем OM имя отображаемого массива
 print *, "Starting Array Viewer"
 ! Запуск OM с использованием fav-подпрограммы
 call favStartViewer(hv, status)
 if(status /= 0) then
  call favGetErrorNo(hv, nError, status)
  if(nError /= 0) then
   print *, "Array Viewer reports error ", nError
   stop
  end if
 end if
```

```
! Передаем ОМ данные подлежащего отображению массива
call favSetArray(hv, fun, status)
! Задаем заголовок экземпляра ОМ
call favSetArrayName(hv, "Bspline(x, y)", status)
! Отображаем массив в виде векторного графа
call favSetGraphType(hv, VectorGraph, status)
! Задаем режим вывода заданных пользователем имен осей координат
call favSetUseAxisLabel(hv, x_axis, 1, status)
call favSetUseAxisLabel(hv, y_axis, 1, status)
call favSetUseAxisLabel(hv, z_axis, 1, status)
! Новые (вместо dim1, dim2 и z) имена x- и y-осей координат
! Длина переменной, задающей имя оси, равна AV_MAX_LABEL_LEN
call favSetAxisLabel(hv, x_axis, xLabel, status)
call favSetAxisLabel(hv, y_axis, yLabel, status)
call favSetAxisLabel(hv, z_axis, zLabel, status)
! Устанавливаем режим явного задания разметок координатных осей
call favSetAxisAutoDetail(hv, 0, status)
! Число больших разметок на координатных осях
call favSetNumMajorTickmarks(hv, x_axis, 3, status)
call favSetNumMajorTickmarks(hv, y_axis, 3, status)
call favSetNumMajorTickmarks(hv, z_axis, 3, status)
! Показываем ОМ на экране
call favShowWindow(hv, av_true, status)
print *, "Press any key to close down the viewer"
key = getcharqq( )
call favEndViewer(hv, status)          ! Закрываем ОМ
call faglEndWatch(fun, status)         ! Освобождаем ресурсы
end subroutine vGraph2
```

Результат:

x	y	s(x, y)	Error
0.0000	0.0000	0.0000	0.00000
0.0000	0.3333	0.0000	0.00000
0.0000	0.6667	0.0000	0.00000
0.0000	1.0000	0.0000	0.00000
0.3333	0.0000	0.0370	0.00000
0.3333	0.3333	0.1481	0.00000
0.3333	0.6667	0.2593	0.00000
0.3333	1.0000	0.3704	0.00000
0.6667	0.0000	0.2963	0.00000
0.6667	0.3333	0.5185	0.00000

0.6667	0.6667	0.7407	0.00000
0.6667	1.0000	0.9630	0.00000
1.0000	0.0000	1.0000	0.00000
1.0000	0.3333	1.3333	0.00000
1.0000	0.6667	1.6667	0.00000
1.0000	1.0000	2.0000	0.00000

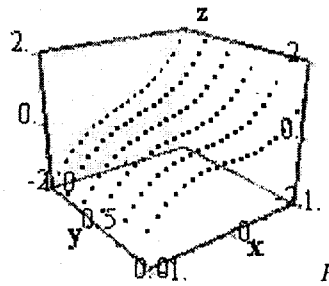


Рис. 1.6. B-сплайн для функции $f(x, y) = x^3 + x * y$

1.4.1.5. Подпрограмма BS3IN (DBS3IN)

Вычисляет трехмерный интерполяционный ТП-B-сплайн, возвращая его коэффициенты. Имеет вызов

CALL BS3IN(*nxdata, xdata, nydata, ydata, nzdata, zdata, fdata, Ldf,* &
mdf, kxord, kyord, kzord, xknot, yknot, zknot, bscoef)

Параметр *bscoef* является *выходным*. Остальные параметры - *входные*. Смысл параметров см. в табл. 1.5.

Комментарий. Информационные ошибки BS3IN аналогичны ошибкам BS2IN.

Описание:

Подпрограмма BS3IN вычисляет на наборе точек (x_i, y_j, z_k, f_{ijk}), где $1 \le i \le n_x$, $1 \le j \le n_y$ и $1 \le k \le n_z$, интерполяционный ТП-B-сплайн в виде

$$\sum_{l=1}^{n_z}\sum_{m=1}^{n_y}\sum_{n=1}^{n_x} b_{nml} B_{nk_x t_x}(x) B_{mk_y t_y}(y) B_{lk_z t_z}(z).$$

Алгоритм требует, чтобы

$$t_x(k_x) \le x_i \le t_x(n_x + 1),\ 1 \le i \le n_x;$$

$$t_y(k_y) \le y_j \le t_y(n_y + 1),\ 1 \le j \le n_y;$$

$$t_z(k_z) \le z_k \le t_z(n_z + 1),\ 1 \le k \le n_z.$$

Коэффициенты ТП-В-сплайна находятся в результате решения линейной системы

$$\sum_{l=1}^{n_z}\sum_{m=1}^{n_y}\sum_{n=1}^{n_x} b_{nml}B_{nk_xt_x}(x)B_{mk_yt_y}(y)B_{lk_zt_z}(z) = f_{ijk}.$$

Подпрограмма BS3IN основана на процедуре SPLI2D, приведенной в [7].

1.4.2. ОЦЕНКА, ИНТЕГРИРОВАНИЕ, ПРЕОБРАЗОВАНИЕ В-СПЛАЙНОВ

1.4.2.1. Функция BSVAL (DBSVAL)

Возвращает оценку В-сплайна в заданной точке *x*. Имеет вызов

result = BSVAL(*x, korder, xknot, ncoef, bscoef*)

Все параметры функции BSVAL являются *входными*.

Результирующая переменная: BSVAL.

Описание параметров см. в табл. 1.4.

Комментарий. При работе с BSVAL могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
4	4	Число повторяемости узлов не может превышать порядок сплайна
4	5	Узлы должны быть упорядочены по неубыванию их значений

Описание:

Функция BSVAL является частным случаем процедуры BSDER, оценивающей либо функцию, либо производную В-сплайна. В точке *x* по известному вектору узлов *t*, числу коэффициентов *n* и вектору коэффициентов *a* функция BSVAL возвращает

$$\sum_{j=1}^{n} a_j B_{jkl}(x).$$

Заметьте, что функция BSVAL рассматривает В-сплайн как функцию, непрерывную справа и слева соответственно около *xknot*(*korder*) и *xknot*(*ncoef* + 1).

Пример употребления BSVAL приведен при рассмотрении BSINT.

1.4.2.2. Функция BSDER (DBSDER)

Возвращает производную одномерного B-сплайна в заданной точке *x*. Имеет вызов

result = BSDER(*ideriv, x, korder, xknot, ncoef, bscoef*)

Все параметры функции BSDER являются *входными*.

Результирующая переменная: BSDER.

Описание параметров см. в табл. 1.4.

Комментарий. Возникают те же, что и в случае BSVAL, информационные ошибки.

Описание:

Функция BSDER основана на процедуре BVALUE, приведенной в [7]. Функция возвращает производную порядка *i*, равную

$$\sum_{j=1}^{n} b_j B_{jkt}^{(i)}(x).$$

Пример. Подпрограммой BSINT вычисляется интерполяционный B-сплайн по точкам, задаваемым функцией

$$f(x) = \sqrt{x}.$$

Затем функция BSDER употребляется для оценки и сплайна, и его первой производной в первой, последней и средней точках каждого подынтервала. В качестве результата выводятся приведенные оценки и соответствующие им ошибки.

```
program bsderTest
 use dfimsl
 integer(4), parameter :: korder = 3, ndata = 5, nknot = ndata + korder
 integer(4) :: i, ncoef
real(4) :: bscoef(ndata), bsder, bt0, bt1, df, f, fdata(ndata), x, xdata(ndata), xknot(nknot), xt
 f(x) = sqrt(x); df(x) = 0.5 / sqrt(x)      ! Задаем функцию и ее производную
 do i = 1, ndata                            ! Задаем точки интерполяции
  xdata(i) = float(i) / float(ndata)
  fdata(i) = f(xdata(i))
 end do                                     ! Генерируем последовательность узлов
 call bsnak(ndata, xdata, korder, xknot)
 ! Выполняем интерполяцию
 call bsint(ndata, xdata, fdata, korder, xknot, bscoef)
 ! Вывод заголовка
 write(*, "(6x, 'x', 8x, 's(x)', 7x, 'Error', 8x, 's''(x)', 8x, 'Error', /)")
```

```
ncoef = ndata                          ! Вывод на сетке
xt = xdata(1)
! Оценка сплайна и его первой производной
bt0 = bsder(0, xt, korder, xknot, ncoef, bscoef)
bt1 = bsder(1, xt, korder, xknot, ncoef, bscoef)
write(*, 99999) xt, bt0, f(xt) - bt0, bt1, df(xt) - bt1
do i = 2, ndata
 xt = (xdata(i - 1) + xdata(i)) / 2.0
 ! Оценка сплайна и его первой производной
 bt0 = bsder(0, xt, korder, xknot, ncoef, bscoef)
 bt1 = bsder(1, xt, korder, xknot, ncoef, bscoef)
 write(*, 99999) xt, bt0, f(xt) - bt0, bt1, df(xt) - bt1
 xt = xdata(i)
 ! Повторяем оценку сплайна и его первой производной
 bt0 = bsder(0, xt, korder, xknot, ncoef, bscoef)
 bt1 = bsder(1, xt, korder, xknot, ncoef, bscoef)
 write(*, 99999) xt, bt0, f(xt) - bt0, bt1, df(xt) - bt1
end do
99999 format(' ', f6.4, 5x, f7.4, 3x, f10.6, 5x, f8.4, 3x, f10.6)
end program bsderTest
```

Результат:

x	s(x)	Error	s'(x)	Error
0.2000	0.4472	0.000000	1.0423	0.075738
0.3000	0.5456	0.002084	0.9262	-0.013339
0.4000	0.6325	0.000000	0.8101	-0.019553
0.5000	0.7077	-0.000557	0.6940	0.013071
0.6000	0.7746	0.000000	0.6446	0.000869
0.7000	0.8366	0.000071	0.5952	0.002394
0.8000	0.8944	0.000000	0.5615	-0.002525
0.9000	0.9489	-0.000214	0.5279	-0.000818
1.0000	1.0000	0.000000	0.4942	0.005814

1.4.2.3. Подпрограмма BS1GD (DBS1GD)

Возвращает производную одномерного B-сплайна на сетке. Имеет вызов

CALL BS1GD(*ideriv, n, xvec, korder, xknot, ncoef, bscoef, value*)

Параметр *value* является *выходным*. Остальные параметры - *входные*. Описание параметров см. в табл. 1.4.

Комментарий. Может возникнуть информационная ошибка типа 4 с кодом 5, означающая, что данные в векторе *xvec* расположены не по возрастанию.

Описание:

Пусть задан вектор x размера n, такой, что $x_i < x_i + 1$ для $i = 1, \ldots, n - 1$; пусть j - порядок производной, а s - ранее вычисленный B-сплайн, представляемый последовательностью узлов и своими коэффициентами. Подпрограмма BS1GD возвращает в векторе *value* значения

$$s^{(j)}x_i,\ i = 1, \ldots, n.$$

Подпрограмму BS1GD можно заменить, вызывая в цикле функцию BSDER. Однако отдельный вызов BS1GD более эффективен. Подпрограмма BS1GD преобразовывает, употребляя BSCPP, B-сплайн в KM-форму и затем использует функцию PPVAL для оценки. Подпрограмма BS1GD основана на процедурах BSPLPP и PPVALU, приведенных в [7].

Пример. Вычисляется такой же, что и в вышеприведенном примере для BSDER, квадратичный, порядка 3 сплайн. Выполняются те же, что и в упомянутом примере, но уже с применением BS1GD, оценки.

```
program bs1gdTest
 use dfimsl
 integer(4), parameter :: korder = 3, ndata = 5, nknot = ndata + korder, nfgrid = 9
 integer(4) :: i, ncoef
 real(4) :: ans0(nfgrid), ans1(nfgrid), bscoef(ndata), fdata(ndata)
 real(4) :: x, xdata(ndata), xknot(nknot), xvec(nfgrid), df, f
 f(x) = sqrt(x)                          ! Задаем функцию и ее производную
 df(x) = 0.5 / sqrt(x)
 do i = 1, ndata                         ! Задаем точки интерполяции
  xdata(i) = float(i) / float(ndata)
  fdata(i) = f(xdata(i))
 end do
 call bsnak(ndata, xdata, korder, xknot) ! Генерируем последовательность узлов
 ! Выполняем интерполяцию
 call bsint(ndata, xdata, fdata, korder, xknot, bscoef)
 write(*, "(6x, 'x', 8x, 's(x)', 7x, 'Error', 8x, 's''(x)', 8x, 'Error', /)")
 ncoef = ndata; xvec(1) = xdata(1)       ! Вывод на сетке
 do i = 2, 2 * ndata - 2, 2
  xvec(i) = (xdata(i / 2 + 1) + xdata(i / 2)) / 2.0
  xvec(i + 1) = xdata(i / 2 + 1)
 end do
 call bs1gd(0, 2 * ndata - 1, xvec, korder, xknot, ncoef, bscoef, ans0)
 call bs1gd(1, 2 * ndata - 1, xvec, korder, xknot, ncoef, bscoef, ans1)
 do i = 1, 2 * ndata - 1
  write(*, 99998) xvec(i), ans0(i), f(xvec(i)) - ans0(i), ans1(i), df(xvec(i)) - ans1(i)
 end do
```

```
99998 format(' ', f6.4, 5x, f7.4, 3x, f10.6, 5x, f8.4, 3x, f10.6)
end program bs1gdTest
```

Результат тот же, что и в примере для BSDER.

1.4.2.4. Функция BSITG (DBSITG)

Возвращает интеграл B-сплайна. Имеет вызов

result = BSITG(*a, b, korder, xknot, ncoef, bscoef*)

Все параметры функции BSITG являются *входными*.

Результирующая переменная: BSITG.

Описание параметров см. в табл. 1.4.

Комментарий. При работе с BSITG могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	7	Верхняя и нижняя границы интегрирования равны
3	8	Нижняя граница интегрирования меньше, чем *xknot*(*korder*)
3	9	Верхняя граница интегрирования больше, чем *xknot*(*ncoef* + 1)
4	4	Число повторяемости узлов не может превышать порядок сплайна
4	5	Узлы должны быть упорядочены по неубыванию их значений

Описание:

Функция BSITG на отрезке [*a, b*] возвращает значение

$$\int_a^b \sum_{i=1}^{n_c} a_i B_{ik_t}(x)dx .$$

Причем $t_1 = \ldots = t_k$ и $t_{n+1} = \ldots = t_{n+k}$.

Пример. Интегрируется сплайн четвертой степени (порядка $k = 5$), воспроизводящий функцию x^3. Результат сравнивается с точным значением.

```
program bsitgTest
 use dfimsl
 integer(4), parameter :: korder = 5, ndata = 21, nknot = ndata + korder
 integer(4) :: i, ncoef
 real(4) :: a, b, bscoef(ndata), bsitg, error, exact,                    &
       f, fdata(ndata), fi, val, x, xdata(ndata), xknot(nknot)
 f(x) = x * x * x; fi(x) = x**4 / 4.0      ! Задаем функцию и интеграл
 do i = 1, ndata                           ! Задаем точки интерполяции
   xdata(i) = float(i - 11) / 10.0
```

```
  fdata(i) = f(xdata(i))
 end do
 ! Генерируем последовательность узлов
 call bsnak(ndata, xdata, korder, xknot)
 ! Находим интерполяционный B-сплайн
 call bsint(ndata, xdata, fdata, korder, xknot, bscoef)
 ncoef = ndata; a = 0.0; b = 1.0          ! Интегрируем от a до b
 val = bsitg(a, b, korder, xknot, ncoef, bscoef)
 exact = fi(b) - fi(a)
 error = exact - val
 write(*, 99999) a, b, val, exact, error  ! Вывод результатов
 99999 format(' On the closed interval (', f3.1, ',', f3.1, ') we have:', /, 1x,    &
       'Computed integral = ', f10.5, /, 1x, 'Exact integral = ',                   &
       f10.5, /, 1x, 'Error ', ' = ', f10.6, /)
end program bsitgTest
```

Результат:

```
On the closed interval (0.0,1.0) we have:
Computed Integral = 0.25000
Exact Integral = 0.25000
Error = 0.000000
```

1.4.2.5. Функция BS2VL (DBS2VL)

Возвращает оценку двумерного ТП-В-сплайна в заданной точке (x, y). Имеет вызов

result = BS2VL(*x, y, kxord, kyord, xknot, yknot, nxcoef, nycoef, bscoef*)

Все параметры функции BS2VL являются *входными.*

Результирующая переменная: BS2VL.

Описание параметров см. в табл. 1.5.

Описание:

Функция BS2VL является частным случаем функции BS2DR, оценивающей частные производные двумерного ТП-В-сплайна. Функция BS2VL для сплайна s, заданного коэффициентами b, в точке (x, y) возвращает

$$s(x,y) = \sum_{m=1}^{n_y} \sum_{n=1}^{n_x} b_{nm} B_{nk_x t_x}(x) B_{mk_y t_y}(y).$$

Пример употребления BS2VL см. в примере для BS2IN.

1.4.2.6. Функция BS2DR (DBS2DR)

Возвращает оценку производной двумерного ТП-В-сплайна в заданной точке (x, y). Имеет вызов

result = BS2DR(*ixder, iyder, x, y, kxord, kyord, xknot, yknot, nxcoef, nycoef, bscoef*)

Все параметры функции BS2DR являются *входными*.

Результирующая переменная: BS2DR.

Описание параметров см. в табл. 1.5.

Комментарий. При работе с BS2DR могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	1	x-координата точки, равная $xvec(i)$, не удовлетворяет неравенству $xknot(kxord) \le x \le xknot(nxcoef + 1)$
3	2	y-координата точки, равная $yvec(i)$, не удовлетворяет неравенству $yknot(kyord) \le y \le yknot(nycoef + 1)$

Описание:

Функция BS2DR возвращает (p, q)-частную производную двумерного ТП-В-сплайна в точке (x, y)

$$s^{(p,q)}(x,y) = \sum_{m=1}^{N_y}\sum_{n=1}^{N_x} b_{nm} B^{(p)}_{nk_x t_x}(x) B^{(q)}_{mk_y t_y}(y).$$

Пример. Первоначально вычисляется ТП-В-сплайн по точкам, принадлежащим функции $f(x, y) = x^4 + x^3y^2$. Затем функция BS2DR употребляется для вычисления частной производной с *ixder* = 2 и *iyder* = 1. Результат сравнивается на сетке 4×4 с истинным значением.

```
program bs2drTest
 use dfimsl
 integer(4), parameter :: kxord = 5, kyord = 3, nxdata = 21, nydata = 6,          &
      Ldf = nxdata, nxknot = nxdata + kxord, nyknot = nydata + kyord
 integer i, j, nxcoef, nycoef
 real bscoef(nxdata,nydata), f, f21, fdata(Ldf, nydata), s21, x, xdata(nxdata),   &
      xknot(nxknot), y, ydata(nydata), yknot(nyknot)
 ! Определяем функцию и ее (2, 1)-частную производную
 f(x, y) = x * x * x * x + x * x * x * y * y
 f21(x, y) = 12.0 * x * y
 ! Задаем точки интерполяции
```

```
do i = 1, nxdata
  xdata(i) = float(i - 11) / 10.0
end do
do i = 1, nydata
  ydata(i) = float(i - 1) / 5.0
end do
! Генерируем последовательность узлов
call bsnak(nxdata, xdata, kxord, xknot)
call bsnak(nydata, ydata, kyord, yknot)
do i = 1, nydata                          ! Генерируем fdata
 do j = 1, nxdata
   fdata(j, i) = f(xdata(j), ydata(i))
  end do
end do                                    ! Решаем задачу интерполяции
call bs2in(nxdata, xdata, nydata, ydata, fdata, Ldf, kxord, kyord, xknot, yknot, bscoef)
nxcoef = nxdata; nycoef = nydata          ! Вывод заголовка
write(*, "(38x, '(2, 1)', /, 13x, 'x', 14x, 'y', 10x, 's(x, y)', 5x, 'Error')")
! Вывод (2, 1)-производной в области [0.0, 1.0]×[0.0, 1.0] на сетке 4×4
do i = 1, 4
 do j = 1, 4
  x = float(i - 1) / 3.0
  y = float(j - 1) / 3.0
  ! Оцениваем сплайн
  s21 = bs2dr(2, 1, x, y, kxord, kyord, xknot, yknot, nxcoef, nycoef, bscoef)
  write(*, '(3f15.4, f15.6)') x, y, s21, f21(x, y) - s21
 end do
end do
end program bs2drTest
```

Результат:

(2, 1)

x	*y*	s(*x*, *y*)	Error
0.0000	0.0000	0.0000	0.000000
0.0000	0.3333	0.0000	0.000000
0.0000	0.6667	0.0000	0.000000
0.0000	1.0000	0.0000	0.000000
0.3333	0.0000	0.0000	-0.000002
0.3333	0.3333	1.3333	0.000001
0.3333	0.6667	2.6667	-0.000001
0.3333	1.0000	4.0000	0.000002
0.6667	0.0000	0.0000	-0.000035

0.6667	0.3333	2.6666	0.000041
0.6667	0.6667	5.3334	-0.000018
0.6667	1.0000	8.0000	0.000026
1.0000	0.0000	0.0000	-0.000019
1.0000	0.3333	3.9998	0.000187
1.0000	0.6667	8.0000	-0.000036
1.0000	1.0000	11.9997	0.000267

1.4.2.7. Подпрограмма BS2GD (DBS2GD)

Оценивает производную двумерного ТП-В-сплайна на сетке. Имеет вызов

CALL BS2GD(*ixder, iyder, nx, xvec, ny, yvec, kxord, kyord,* &
xknot, yknot, nxcoef, nycoef, bscoef, value, Ldvalu)

Параметр *value* является *выходным*. Остальные параметры - *входные*. Описание параметров см. в табл. 1.5.

Комментарий. При работе с BS2GD могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	1	*x*-координата точки, равная *xvec*(*i*), не удовлетворяет неравенству *xknot*(*kxord*) ≤ *x* ≤ *xknot*(*nxcoef* + 1)
3	2	*y*-координата точки, равная *yvec*(*i*), не удовлетворяет неравенству *yknot*(*kyord*) ≤ *y* ≤ *yknot*(*nycoef* + 1)
4	3	Вектор *xvec* не является строго возрастающим
4	4	Вектор *yvec* не является строго возрастающим

Описание см. в предшествующем разделе.

Пример. Решается та же, что и в примере для функции BS2DR, задача с тем отличием, что значения частных производных вычисляются на сетке 4×4.

```
program bs2gdTest
 use dfimsl
 integer(4) :: i, j, kxord, kyord, Ldf, nxcoef, nxdata, nycoef, nydata
 real(4) :: fdata(21, 6), bscoef(21, 6), xdata(21), xknot(26),                &
      ydata(6), yknot(9), f, f21, value(4, 4), x, xvec(4), y, yvec(4)
 ! Определяем функцию и ее (2, 1)-частную производную
 f(x, y) = x * x * x * x + x * x * x * y * y
 f21(x, y) = 12.0 * x * y
 kxord = 5; kyord = 3
 nxdata = 21; nydata = 6
```

```
 Ldf = nxdata
 ! Задаем точки интерполяции
 do i = 1, nxdata
   xdata(i) = float(i - 11) / 10.0
 end do
 do i = 1, nydata
   ydata(i) = float(i - 1) / 5.0
 end do
 ! Генерируем последовательность узлов
 call bsnak(nxdata, xdata, kxord, xknot)
 call bsnak(nydata, ydata, kyord, yknot)
 do i = 1, nydata                          ! Генерируем fdata
  do j = 1, nxdata
    fdata(j, i) = f(xdata(j), ydata(i))
  end do
 end do                                    ! Решаем задачу интерполяции
 call bs2in(nxdata, xdata, nydata, ydata, fdata, Ldf, kxord, kyord, xknot, yknot, bscoef)
 nxcoef = nxdata; nycoef = nydata          ! Вывод заголовка
 write(*, "(38x, '(2, 1)', /, 13x, 'x', 14x, 'y', 10x, 's(x, y)', 5x, 'Error')")
 ! Вывод (2, 1)-производной в области [0.0, 1.0]×[0.0, 1.0] на сетке 4×4
 do i = 1, 4
   xvec(i) = float(i - 1) / 3.0
   yvec(i) = xvec(i)
 end do
 call bs2gd(2, 1, 4, xvec, 4, yvec, kxord, kyord,                              &
       xknot, yknot, nxcoef, nycoef, bscoef, value, 4)
 do i = 1, 4
  do j = 1, 4
   write(*, '(3f15.4, f15.6)') xvec(i), yvec(j), value(i, j), f21(xvec(i), yvec(j)) - value(i, j)
  end do
 end do
end program bs2gdTest
```

Результат тот же, что и для функции BS2DR.

1.4.2.8. Функция BS2IG (DBS2IG)

Возвращает интеграл двумерного ТП-В-сплайна в прямоугольной области. Имеет вызов

result = BS2IG(*a, b, c, d, kxord, kyord, xknot, yknot, nxcoef, nycoef, bscoef*)

Все параметры функции BS2IG являются *входными*.

Результирующая переменная: BS2IG.

Описание параметров см. в табл. 1.5.

Комментарий. При работе с BS2IG могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	1	Нижняя граница интегрирования по *x* меньше, чем *xknot*(*kxord*)
3	2	Верхняя граница интегрирования по *x* больше, чем *xknot*(*nxcoef* + 1)
3	3	Нижняя граница интегрирования по *y* меньше, чем *yknot*(*kyord*)
3	4	Верхняя граница интегрирования по *y* больше, чем *yknot*(*nycoef* + 1)
4	13	Число повторяемости узлов не может превышать порядок сплайна
4	14	Узлы должны быть упорядочены по неубыванию их значений

Описание:

Функция BS2IG в области $[a, b]\times[c, d]$ возвращает

$$\int_a^b \int_c^d \sum_{i=1}^{n_{cx}} \sum_{j=1}^{n_{cy}} b_{ij} B_{ij}\, dxdy ,$$

где

$$B_{ij}(x, y) = B_{ik_x t_x}(x) B_{jk_y t_{xy}}(y) .$$

Причем $t_1 = \ldots = t_k$ и $t_{n+1} = \ldots = t_{n+k}$, где k - порядок сплайна в направлении x или y.

Пример. Интегрируется интерполяционный ТП-В-сплайн с $kx = 5$ и $ky = 2$, построенный на точках, задаваемых функцией $x^3 + xy$. Интегрирование выполняется в области $[0, 1]\times[0.5, 1]$. Точный ответ известен, он равен 5/16.

```
program bs2igTest
 use dfimsl
 integer(4), parameter :: kxord = 5, kyord = 2, nxdata = 21, nydata = 6,          &
      Ldf = nxdata, nxknot = nxdata + kxord, nyknot = nydata + kyord
 integer(4) :: i, j, nxcoef, nycoef
 real(4) :: a, b, bs2ig, bscoef(nxdata, nydata), c, d, f, fdata(Ldf, nydata), fi, val,  &
      x, xdata(nxdata), xknot(nxknot), y, ydata(nydata), yknot(nyknot)
 f(x, y) = x * x * x + x * y                  ! Задаем функцию и интеграл
 fi(a, b, c, d) = .25 * ((b**4 - a**4) * (d - c) + (b * b - a * a) * (d * d - c * c))
 ! Задаем точки интерполяции и генерируем последовательность узлов
 do i = 1, nxdata
   xdata(i) = float( i -11) / 10.0
 end do
 do i = 1, nydata
```

```
  ydata(i) = float(i - 1) / 5.0
 end do
 call bsnak(nxdata, xdata, kxord, xknot)
 call bsnak(nydata, ydata, kyord, yknot)
 do i = 1, nydata                          ! Генерируем fdata
  do j = 1, nxdata
   fdata(j, i) = f(xdata(j), ydata(i))
  end do
 end do                                    ! Вычисляем сплайн
 call bs2in(nxdata, xdata, nydata, ydata, fdata, Ldf, kxord, kyord, xknot, yknot, bscoef)
 ! Интегрируем в прямоугольной области [0.0, 1.0]×[0.0, 0.5]
 nxcoef = nxdata; nycoef = nydata
 a = 0.0; b = 1.0; c = 0.5; d = 1.0
 val = bs2ig(a, b, c, d, kxord, kyord, xknot, yknot, nxcoef, nycoef, bscoef)
 write(*, 99999) val, fi(a, b, c, d), fi(a, b, c, d) - val
 99999 format(' Computed integral = ', f10.5, /, ' Exact integral ', '= ',          &
       f10.5, /, ' Error ', '= ', f10.6, /)
end program bs2igTest
```

Результат:

```
Computed integral = 0.31250
Exact integral = 0.31250
Error = 0.000000
```

1.4.2.9. Функция BS3VL (DBS3VL)

Возвращает оценку трехмерного ТП-В-сплайна в заданной точке (x, y, z). Имеет вызов

result = BS3VL(*x, y, z, kxord, kyord, kzord, xknot, yknot, zknot,* &
nxcoef, nycoef, nzcoef, bscoef)

Все параметры функции BS3VL являются *входными.*

Результирующая переменная: BS3VL.

Описание параметров см. в табл. 1.5.

Описание:

Функция BS3VL является частным случаем функции BS3DR, оценивающей частные производные трехмерного ТП-В-сплайна. Функция BS3VL для сплайна s, заданного коэффициентами b, в точке (x, y, z) возвращает

$$s(x, y, z) = \sum_{l=1}^{n_z} \sum_{m=1}^{n_y} \sum_{n=1}^{n_x} b_{nm} B_{nk_x t_x}(x) B_{mk_y t_y}(y) B_{lk_z t_z}(z).$$

1.4.2.10. Функция BS3DR (DBS3DR)

Возвращает оценку производной трехмерного ТП-В-сплайна в заданной точке (*x, y*). Имеет вызов

result = BS3DR(*ixder, iyder, izder, x, y, z, kxord, kyord, kzord,* &
xknot, yknot, zknot, nxcoef, nycoef, nzcoef, bscoef)

Все параметры функции BS3DR являются *входными*.

Результирующая переменная: BS3DR.

Описание параметров см. в табл. 1.5.

Комментарий. При работе с BS3DR могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	1	*x*-координата точки, равная *xvec*(*i*), не удовлетворяет неравенству $xknot(kxord) \le x \le xknot(nxcoef+1)$
3	2	*y*-координата точки, равная *yvec*(*i*), не удовлетворяет неравенству $yknot(kyord) \le y \le yknot(nycoef+1)$
3	3	*z*-координата точки, равная *zvec*(*i*), не удовлетворяет неравенству $zknot(kzord) \le y \le zknot(nzcoef+1)$

Описание:

Функция BS3DR возвращает (*p, q, r*)-частную производную двумерного ТП-В-сплайна в точке (*x, y, z*)

$$s^{(p,q,r)}(x,y)=\sum_{l=1}^{n_z}\sum_{m=1}^{n_y}\sum_{n=1}^{n_x} b_{nm}B^{(p)}_{nk_x t_x}(x)B^{(q)}_{mk_y t_y}(y)B^{(r)}_{lk_z t_z}(z).$$

1.4.2.11. Подпрограмма BS3GD (DBS3GD)

Оценивает производную двумерного ТП-В-сплайна на сетке. Имеет вызов

CALL BS3GD(*ixder, iyder, izder, nx, xvec, ny, yvec, nz, zvec,* &
kxord, kyord, kzord, xknot, yknot, zknot, nxcoef, &
nycoef, nzcoef, bscoef, value, Ldvalu, mdvalu)

Параметр *value* является *выходным*. Остальные параметры - *входные*. Описание параметров см. в табл. 1.5.

Комментарий. При работе с BS3GD могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	1	*x*-координата точки, равная *xvec*(*i*), не удовлетворяет неравенству *xknot*(*kxord*) ≤ *x* ≤ *xknot*(*nxcoef* + 1)
3	2	*y*-координата точки, равная *yvec*(*i*), не удовлетворяет неравенству *yknot*(*kyord*) ≤ *y* ≤ *yknot*(*nycoef* + 1)
3	3	*z*-координата точки, равная *zvec*(*i*), не удовлетворяет неравенству *zknot*(*kzord*) ≤ *y* ≤ *zknot*(*nzcoef* + 1)
4	3	Вектор *xvec* не является строго возрастающим
4	4	Вектор *yvec* не является строго возрастающим
4	5	Вектор *zvec* не является строго возрастающим

Описание см. в предшествующем разделе.

1.4.2.12. Функция BS3IG (DBS3IG)

Возвращает интеграл трехмерного ТП-В-сплайна в прямоугольной области. Имеет вызов

result = BS3IG(*a, b, c, d, e, f, kxord, kyord, kzord,* &
xknot, yknot, zknot, nxcoef, nycoef, nzcoef, bscoef)

Все параметры функции BS3IG являются *входными*.

Результирующая переменная: BS3IG.

Описание параметров см. в табл. 1.5.

Комментарий. При работе с BS3IG могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	1	Нижняя граница интегрирования по *x* меньше, чем *xknot*(*kxord*)
3	2	Верхняя граница интегрирования по *x* больше, чем *xknot*(*nxcoef* + 1)
3	3	Нижняя граница интегрирования по *y* меньше, чем *yknot*(*kyord*)
3	4	Верхняя граница интегрирования по *y* больше, чем *yknot*(*nycoef* + 1)
3	5	Нижняя граница интегрирования по *y* меньше, чем *zknot*(*kzord*)
3	6	Верхняя граница интегрирования по *y* больше, чем *zknot*(*nzcoef* + 1)
4	13	Число повторяемости узлов не может превышать порядок сплайна
4	14	Узлы должны быть упорядочены по неубыванию их значений

Описание:

Функция BS3IG в трехмерной области [*a, b*]×[*c, d*]×[*e, f*] вычисляет определенный интеграл

$$\int_a^b \int_c^d \int_e^f \sum_{i=1}^{n_{cx}} \sum_{j=1}^{n_{cy}} \sum_{m=1}^{n_{cz}} b_{ijm} B_{ijm} dxdydz ,$$

где

$$B_{ijm}(x, y, z) = B_{ik_x t_x}(x) B_{jk_y t_y}(y) B_{mk_z t_z}(z) .$$

Причем $t_1 = \ldots = t_k$ и $t_{n+1} = \ldots = t_{n+k}$, где k - порядок сплайна в направлении x, y или z.

Пример. Оценивается интеграл трехмерного B-сплайна, построенного на точках, принадлежащих функции $x^3 + xyz$. Порядок сплайна $k_x = 5$, $k_y = 2$, $k_z = 3$. Интегрирование выполняется в трехмерной области $[0, 1]\times[0.5, 1]\times[0, 0.5]$. Точный ответ равен 11/128.

```
program bs3inTest
 use dfimsl
 integer(4), parameter :: kxord = 5, kyord = 2, kzord = 3, nxdata = 21, nydata = 6,    &
      nzdata = 8, Ldf = nxdata, mdf = nydata, nxknot = nxdata + kxord,                 &
      nyknot = nydata + kyord, nzknot = nzdata + kzord
 integer(4) :: i, j, k, nxcoef, nycoef, nzcoef
 real(4) :: a, b, bs3ig, bscoef(nxdata,nydata,nzdata), c, d, e, f,                    &
      fdata(Ldf, mdf, nzdata), ff, fig, g, h, ri, rj, val, x, xdata(nxdata),          &
      xknot(nxknot), y, ydata(nydata), yknot(nyknot), z, zdata(nzdata), zknot(nzknot)
 f(x, y, z) = x * x * x + x *y * z          ! Задаем функцию
 ! Генерируем точки интерполяции и затем узлы
 do i = 1, nxdata
   xdata(i) = float(i - 11) / 10.0
 end do
 do i = 1, nydata
   ydata(i) = float(i - 1) / float(nydata - 1)
 end do
 do i = 1, nzdata
   zdata(i) = float(i - 1) / float(nzdata - 1)
 end do
 call bsnak(nxdata, xdata, kxord, xknot)
 call bsnak(nydata, ydata, kyord, yknot)
 call bsnak(nzdata, zdata, kzord, zknot)
 do k = 1, nzdata                          ! Генерируем значения функции fdata
  do j = 1, nydata
   do i = 1, nxdata
     fdata(i, j, k) = f(xdata(i), ydata(j), zdata(k))
   end do
  end do
```

```
end do                                    ! Вычисляем интерполяционный B-сплайн
call bs3in(nxdata, xdata, nydata, ydata, nzdata, zdata, fdata, Ldf, mdf,    &
     kxord, kyord, kzord, xknot, yknot, zknot, bscoef)
nxcoef = nxdata; nycoef = nydata; nzcoef = nzdata
a = 0.0; b = 1.0; c = 0.5; d = 1.0; e = 0.0; ff = 0.5
! Выполняем оценку определенного интеграла
val = bs3ig(a, b, c, d, e, ff, kxord, kyord, kzord,                         &
     xknot, yknot, zknot, nxcoef, nycoef, nzcoef, bscoef)
! Выполняем прямое интегрирование
g = 0.5 * (b**4 - a**4)
h = (b - a) * (b + a)
ri = g * (d - c)
rj = 0.5 * h * (d - c) * (d + c)
fig = 0.5 * (ri * (ff - e) + 0.5 * rj * (ff - e) * (ff + e))
! Выводим результат
write(*, 1) val, fig, fig - val
1 format (' Computed integral = ', f10.5, /,                                &
     ' Exact integral ', '= ', f10.5,/, ' Error ', '= ', f10.6,/)
end program bs3inTest
```

Результат:

```
Computed integral = 0.08594
Exact integral = 0.08594
Error = 0.000000
```

1.4.2.13. Подпрограмма BSCPP (DBSCPP)

Преобразовывает одномерный сплайн из B-сплайнового представления в КМ-представление. Имеет вызов

CALL BSCPP(*korder, xknot, ncoef, bscoef, nppcf, break, ppcoef*)

Параметры подпрограммы BSCPP:

Входные: korder, xknot, ncoef, bscoef.

Выходные: nppcf, break, ppcoef.

Описание параметров см. в табл. 1.4.

Комментарий. При работе с BSCPP могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
4	4	Число повторяемости узлов не может превышать порядок сплайна
4	5	Узлы должны быть упорядочены по неубыванию их значений

Описание:

Подпрограмма BSCPP основана на процедуре BSPLPP, приведенной в [7]. Получаемое в результате ее применения КМ-представление одномерного В-сплайна используется в том числе и для более эффективной его оценки. Затраты на преобразование окупаются, если для каждого куска из КМ-представления В-сплайна необходимо получить 3 или более значения функции.

Пример употребления BSCPP см. в разд. 1.5.3.

1.5. ОЦЕНКА КУСОЧНО-МНОГОЧЛЕННЫХ СПЛАЙНОВ

1.5.1. ПЕРЕЧЕНЬ И ПАРАМЕТРЫ ПРОЦЕДУР

Перечень процедур приведен в табл. 1.6. Их параметры - в ранее сформированной табл. 1.4.

Таблица 1.6. Процедуры, оценивающие КМ-представление одномерного сплайна

Процедура	*Описание*
PPVAL	Возвращает оценку КМ-представления одномерного сплайна в заданной точке
PPDER	Возвращает производную КМ-представления одномерного сплайна в заданной точке
PP1GD	Оценивает производную КМ-представления одномерного сплайна на сетке
PPITG	Возвращает интеграл одномерного сплайна, используя его КМ-представление

1.5.2. ФУНКЦИЯ PPVAL (DPPVAL)

Возвращает оценку КМ-представления одномерного сплайна в заданной точке *x*. Имеет вызов

result = PPVAL(*x, korder, nintv, break, ppcoef*)

Все параметры функции PPVAL являются *входными.*

Результирующая переменная: PPVAL.

Описание параметров см. в табл. 1.4.

Описание:

Функция PPVAL основана на процедуре PPVALU, приведенной в [7], и является частным случаем функции PPDER, оценивающей КМ-сплайн или его производную.

Пример употребления PPVAL легко сконструировать, заменив в примере для PPDER вызов

s = ppder(0, x, korder, nppcf, break, ppcoef)

на вызов

s = ppval(x, korder, nppcf, break, ppcoef)

1.5.3. ФУНКЦИЯ PPDER (DPPDER)

Возвращает производную КМ-представления одномерного сплайна в заданной точке *x*. Имеет вызов

result = PPDER(*ideriv, x, korder, nintv, break, ppcoef*)

Все параметры функции PPDER являются *входными*.

Результирующая переменная: PPDER.

Описание параметров см. в табл. 1.4.

Описание:

Функция PPDER основана на процедуре PPVALU, приведенной в [7]. Если точки разрыва находятся в векторе ξ размера $n = nintv + 1$, а коэффициенты КМ-представления сплайна - в векторе c, то значение j-й производной ($j = 0, \ldots, k - 1$) функции f в точке x на интервале $[\xi_i, \xi_{i+1})$ равно

$$f^{(j)}(x) = \sum_{m=j}^{k-1} c_{m+1,i} \frac{(x-\xi_i)^{m-j}}{(m-j)!}.$$

В приведенном представлении функция должна быть непрерывной справа. Если $x < \xi_1$, то в вышеприведенной формуле устанавливается $i = 1$; если $x \geq \xi_n$, то устанавливается $i = n - 1$. Это позволяет расширить КМ-представление сплайна за счет экстраполяции его первого и последнего кусков.

Пример. По точкам, представляющим функцию $f(x) = xe^x$, вычисляется B-сплайн и преобразовывается затем в КМ-представление, используя которое функция PPDER вычисляет оценки сплайна и его первой производной.

```
program ppderTest
 use dfimsl
 integer(4), parameter :: korder = 4, ncoef = 20, ndata = 20, nknot = ndata + korder
 integer(4) :: i, nppcf
 real(4) :: break(ncoef), bscoef(ncoef), df, ds, f, fdata(ndata),                    &
     ppcoef(korder, ncoef), s, x, xdata(ndata), xknot(nknot)
 ! Задаем функцию и ее производную
```

```
f(x) = x * exp(x)
df(x) = (x + 1.0) * exp(x)
do i = 1, ndata                          ! Задаем точки интерполяции
  xdata(i) = float( i - 1) / float(ndata - 1)
  fdata(i) = f(xdata(i))
end do
! Генерируем последовательность узлов
! и вычисляем затем интерполяционный B-сплайн
call bsnak(ndata, xdata, korder, xknot)
call bsint(ncoef, xdata, fdata, korder, xknot, bscoef)
! Преобразовываем B-сплайн в КМ-представление
call bscpp(korder, xknot, ncoef, bscoef, nppcf, break, ppcoef)
! Вывод заголовка таблицы результатов
write(*, "(11x, 'x', 8x, 's(x)', 7x, 'Error', 7x, 's"(x)', 7x, 'Error')")
do i = 1, ndata                          ! Вывод результата
 x = float(i - 1) / float(ndata - 1)
 ! Вычисляем значение сплайна, используя его КМ-представление, в точке x
 s = ppder(0, x, korder, nppcf, break, ppcoef)
 ! Вычисляем теперь значение первой производной сплайна в той же точке x
 ds = ppder(1, x, korder, nppcf, break, ppcoef)
 write(*, '(2f12.3, f12.6, f12.3, f12.6)') x, s, f(x) - s, ds, df(x) - ds
end do
end program ppderTest
```

Результат (для экономии места приводятся первые 5 строк результата):

x	s(*x*)	Error	s'(*x*)	Error
0.000	0.000	0.000000	1.000	-0.000112
0.053	0.055	0.000000	1.109	0.000030
0.105	0.117	0.000000	1.228	-0.000008
0.158	0.185	0.000000	1.356	0.000002
0.211	0.260	0.000000	1.494	0.000000

1.5.4. ПОДПРОГРАММА PP1GD (DPP1GD)

Оценивает производную КМ-представления одномерного сплайна на сетке. Имеет вызов

CALL PP1GD(*ideriv, n, xvec, korder, nintv, break, ppcoef, value*)

Параметр *value* является *выходным*. Остальные параметры - *входные*. Описание параметров см. в табл. 1.4.

Комментарий. Может возникнуть информационная ошибка типа 4 с кодом 4, означающая, что данные в векторе *xvec* расположены не по возрастанию.

Описание:

Подпрограмма PP1GD, получив вектор с абсциссами $x(1{:}n)$, в котором $x_i < x_{i+1}$ для $i = 1, \ldots, n - 1$, вычисляет производную порядка j $f^{(j)}(x_i)$, $i = 1, \ldots, n$ КМ-представления сплайна, возвращая результат в векторе *value*. Функционально подпрограмма PP1GD аналогична программе, в которой функция PPDER вызывается в цикле. Однако PP1GD значительно эффективнее.

Подпрограмма PP1GD основывается на процедуре PPVALU, приведенной в [7].

Пример. Решается та же, что и для вышерассмотренной функции PPDER, задача.

```
program pp1gdTest
 use dfimsl
 integer(4), parameter :: korder = 4, n = 20, ncoef = 20, ndata = 20, nknot = ndata + korder
 integer(4) :: i, nintv, nppcf
 real(4) :: break(ncoef), bscoef(ncoef), df, f, fdata(ndata), ppcoef(korder, ncoef),        &
       value1(n), value2(n), x, xdata(ndata), xknot(nknot), xvec(n)
 ! Задаем функцию и ее производную
 f(x) = x * exp(x)
 df(x) = (x + 1.0) * exp(x)
 do i = 1, ndata                              ! Задаем точки интерполяции
  xdata(i) = float( i - 1) / float(ndata - 1)
  fdata(i) = f(xdata(i))
 end do
 ! Генерируем последовательность узлов
 ! и вычисляем затем интерполяционный B-сплайн
 call bsnak(ndata, xdata, korder, xknot)
 call bsint(ncoef, xdata, fdata, korder, xknot, bscoef)
 ! Преобразовываем B-сплайн в КМ-представление
 call bscpp(korder, xknot, ncoef, bscoef, nppcf, break, ppcoef)
 ! Генерируем оценочные точки
 do i = 1, n
   xvec(i) = float(i - 1) / float(n - 1)
 end do
 nintv = nppcf                                ! Оценка КМ-представления
 call pp1gd(0, n, xvec, korder, nintv, break, ppcoef, value1)
 ! Оценка первой производной КМ-представления B-сплайна
 call pp1gd (1, n, xvec, korder, nintv, break, ppcoef, value2)
 ! Вывод заголовка таблицы результатов
 write(*, "(11x, 'x', 8x, 's(x)', 7x, 'Error', 7x, 's''(x)', 7x, 'Error')")
 do i = 1, n                                  ! Вывод результата на однородной сетке
```

```
 write(*, "(' ', 2f12.3, f12.6, f12.3, f12.6)") xvec(i), value1(i),                &
     f(xvec(i)) - value1(i), value2(i), df(xvec(i)) - value2(i)
end do
end program pp1gdTest
```

Результат тот же, что и в примере для функции PPDER.

1.5.5. ФУНКЦИЯ PPITG (DPPITG)

Возвращает интеграл одномерного сплайна, используя его КМ-представление. Имеет вызов

result = PPITG(*a, b, korder, nintv, break, ppcoef*)

Все параметры функции PPITG являются *входными*.

Результирующая переменная: PPITG.

Описание параметров см. в табл. 1.4.

Пример. Оценивается интеграл квадратичного B-сплайна, полученного по точкам функции $f(x) = x^2$. Для вычисления интеграла функции PPITG передается КМ-представление сплайна. Интегрирование выполняется на отрезках [0, 1/2] и [0, 2]. Точные интегралы на этих отрезках функции $f(x) = x^2$ соответственно равны 1/24 и 8/3.

```
program ppitgTest
 use dfimsl
 integer(4), parameter :: korder = 3, ndata = 10, nknot = ndata + korder
 integer(4) :: i, nppcf
 real(4) :: a, b, break(ndata), bscoef(ndata), exact, f, fdata(ndata), fi,        &
     ppcoef(korder, ndata), value, x, xdata(ndata), xknot(nknot)
 f(x) = x * x                              ! Функция и ее интеграл
 fi(x) = x * x * x / 3.0
 do i = 1, ndata                           ! Точки интерполяции
  xdata(i) = float(i - 1) / float(ndata - 1)
  fdata(i) = f(xdata(i))
 end do
 ! Генерируем последовательность узлов, выполняем интерполяцию
 call bsnak(ndata, xdata, korder, xknot)
 call bsint(ndata, xdata, fdata, korder, xknot, bscoef)
 ! Находим КМ-представление B-сплайна
 call bscpp(korder, xknot, ndata, bscoef, nppcf, break, ppcoef)
 ! Вычисляем интеграл на отрезке [0.0, 0.5]
 a = 0.0; b = 0.5
 value = ppitg(a, b, korder, nppcf, break, ppcoef)
 exact = fi(b) - fi(a)                     ! Точное значение интеграла
```

```
write(*, 1) a, b, value, exact, exact - value
! Вычисляем интеграл на отрезке [0.0, 0.5]
a = 0.0; b = 2.0
value = ppitg(a, b, korder, nppcf, break, ppcoef)
exact = fi(b) - fi(a)                    ! Точное значение интеграла
write(*, 1) a, b, value, exact, exact - value
1 format(' On the closed interval (', f3.1, ',', f3.1, ') we have:', /, 1x,          &
      'Computed integral = ', f10.5, /, 1x, 'Exact integral = ', f10.5, /, 1x, 'Error ', ' = ', f10.6, //)
end program ppitgTest
```

Результат:

```
On the closed interval (0.0,0.5) we have:
Computed integral = 0.04167
Exact integral = 0.04167
Error = 0.000000
On the closed interval (0.0,2.0) we have:
Computed integral = 2.66667
Exact integral = 2.66667
Error = 0.000000
```

1.6. КВАДРАТИЧНЫЕ СПЛАЙНЫ И ИХ ОЦЕНКА

1.6.1. ПЕРЕЧЕНЬ И ПАРАМЕТРЫ ФУНКЦИЙ

Перечень функций приведен в табл. 1.7. Их параметры - в ранее созданных табл. 1.4 и 1.5.

Таблица 1.7. Функции, оценивающие квадратичные сплайны

Процедура	*Описание*
QDVAL	Оценивает, используя квадратичную интерполяцию, в заданной точке x функцию, определенную на наборе точек
QDDER	Оценивает, используя квадратичную интерполяцию, в заданной точке x производную функции, определенной на наборе точек
QD2VL	Оценивает, используя квадратичную интерполяцию, в заданной точке (x, y) функцию, определенную на прямоугольной двумерной сетке
QD2DR	Оценивает в заданной точке (x, y) частную производную функции, определенной на прямоугольной двумерной сетке, используя квадратичную интерполяцию
QD3VL	Оценивает в заданной точке (x, y, z) функцию, определенную на прямоугольной трехмерной сетке, используя квадратичную интерполяцию
QD3DR	Оценивает в заданной точке (x, y, z) частную производную функции, определенной на прямоугольной трехмерной сетке, используя квадратичную интерполяцию

1.6.2. ФУНКЦИЯ QDVAL (DQDVAL)

Оценивает в заданной точке x функцию, определенную на наборе точек, используя квадратичную интерполяцию. Имеет вызов

result = QDVAL(*x*, *ndata*, *xdata*, *fdata*, *check*)

Все параметры функции QDVAL являются *входными*.

Результирующая переменная: QDVAL.

Описание параметров см. в табл. 1.4.

Комментарий. Может возникнуть информационная ошибка типа 4 с кодом 3, означающая, что данные в векторе *xdata* не являются строго возрастающими.

Описание:

Функция QDVAL выполняет квадратичную интерполяцию функции, заданной таблицей данных (x_i, f_i) для $i = 1, \ldots, n$, и возвращает значение квадратичного сплайна в заданной точке. Механизм получения значения таков:

- первоначально находится ближайшая к x табличная точка x_i;
- затем по трем точкам (x_{i-1}, f_{i-1}), (x_i, f_i) и (x_{i+1}, f_{i+1}) находится квадратичный сплайн q;
- функция QDVAL возвращает значение $q(x)$.

Следует заметить, что в общем случае функция QDVAL не является непрерывной в точках сетки.

Пример. Функция QDVAL возвращает значение квадратичного сплайна, аппроксимирующего функцию sin$x$, в точке $\pi/4$. Вычисление интерполяционного квадратичного сплайна выполняется по таблице из 33 равномерно размещенных точек.

```
program qdvalTest
 use dfimsl
 integer(4), parameter :: ndata = 33
 integer(4) :: i
 real(4) :: f, fdata(ndata), h, pi, qt, x, xdata(ndata)
 logical(4) :: check = .true.          ! Проверяем xdata
 f(x) = sin(x)                         ! Задаем функцию и генерируем точки
 xdata(1) = 0.0
 fdata(1) = f(xdata(1))
 h = 1.0/32.0
 do i = 2, ndata
  xdata(i) = xdata(i - 1) + h
```

```
 fdata(i) = f(xdata(i))
 end do
 pi = const('pi'); x = pi / 4.0              ! Получаем π и устанавливаем x
 qt = qdval(x, ndata, xdata, fdata, check) ! Оцениваем функцию в точке π / 4
 write(*, 1) x, f(x), qt, (f(x) - qt)        ! Вывод результата
 1 format (15x, 'x', 6x, 'f(x)', 6x, 'qdval', 5x, 'Error', //, 6x, 4f10.3, /)
end program qdvalTest
```

Результат:

x	f(x)	qdval	Error
0.785	0.707	0.707	0.000

1.6.3. ФУНКЦИЯ QDDER (DQDDER)

Оценивает в заданной точке *x* производную функции, определенной на наборе точек, используя квадратичную интерполяцию. Имеет вызов

result = QDDER(*ideriv, x, ndata, xdata, fdata, check*)

Все параметры функции QDDER являются *входными*.

Результирующая переменная: QDDER.

Описание параметров см. в табл. 1.4.

Комментарии:

1. Может возникнуть информационная ошибка типа 4 с кодом 3, означающая, что данные в векторе *xdata* не являются строго возрастающими.
2. Поскольку интерполяция выполняется квадратичным многочленом, то при *ideriv* > 2 функция QDDER вернет нуль.

Описание функции QDDER аналогично описанию функции QDVAL.

Пример. Функцией QDDER в точке π/4 оценивается значение заданной в виде таблицы функции sin*x*, а также ее первая и вторая производные. Таблица содержит 33 пары (x_i, f_i). Код, связанный с объявлением переменных и генерацией таблицы данных, опускается.

```
. . .
f(x) = sin(x)                              ! Функция и ее производные
f1(x) = cos(x)                             ! Первая производная
f2(x) = -sin(x)                            ! Вторая производная
. . .   ! Подготовка данных
check = .true.                             ! Выполняем проверку данных
write(*, "(33x, 'ider', /, 15x, 'x', 6x, 'ider', 6x, 'f (x)', 5x,'qdder', 6x, 'Error', //)")
ideriv = 0                                 ! Оценка квадратичного сплайна в точке π/4
```

```
qt = qdder(ideriv, x, ndata, xdata, fdata, check)
write(*,"(7x, f10.3, i8, 3f12.3/)") x, ideriv, f(x), qt, (f(x)-qt)
check = .false.                ! Проверка данных не выполняется
ideriv = 1                     ! Оценка первой производной квадратичного сплайна
qt = qdder(ideriv, x, ndata, xdata, fdata, check)
write(*, "(7x, f10.3, i8, 3f12.3/)") x, ideriv, f1(x), qt, (f1(x)-qt)
ideriv = 2                     ! Оценка второй производной квадратичного сплайна
qt = qdder(ideriv, x, ndata, xdata, fdata, check)
write(*, "(7x, f10.3, i8, 3f12.3/)") x, ideriv, f2(x), qt, (f2(x) - qt)
```

Результат:

	ider			
x	ider	f(x)	qdder	error
0.785	0	0.707	0.707	0.000
0.785	1	0.707	0.707	0.000
0.785	2	-0.707	-0.704	-0.003

1.6.4. ФУНКЦИЯ QD2VL (DQD2VL)

Оценивает в заданной точке (x, y) функцию, определенную на прямоугольной двумерной сетке, используя квадратичную интерполяцию. Имеет вызов

result = QD2VL(*x, y, nxdata, xdata, nydata, ydata, fdata, Ldf, check*)

Все параметры функции QD2VL являются *входными.*

Результирующая переменная: QD2VL.

Описание параметров см. в табл. 1.5.

Комментарий. При работе с QD2VL могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
4	6	Вектор *xdata* не является строго возрастающим
4	7	Вектор *ydata* не является строго возрастающим

Описание:

Функция QD2VL выполняет квадратичную интерполяцию функции, заданную таблицей данных (x_i, y_j, f_{ij}) для $i = 1, \ldots, n_x$ и $j = 1, \ldots, n_y$, и возвращает значение квадратичного сплайна в заданной точке. Квадратичный сплайн q строится по шести точкам. Первая - это ближайшая к (x, y) точка сетки. Остальные пять - это точки (x_i, y_j), $(x_{i\pm1}, y_j)$ и $(x_i, y_{j\pm1})$. Функция QD2VL вернет значение $q(x, y)$. Следует заметить, что в общем случае функция QD2VL не является непрерывной в точках сетки.

Пример. По таблице, содержащей 21×42 точек в прямоугольнике [0, 2]×[0, 2], выполняется квадратичная аппроксимация функции $\sin(x + y)$ и ее оценка в точке $x = y = \pi/4$.

```
program qd2vlTest
 use dfimsl
 integer(4), parameter :: nxdata = 21, nydata = 42, Ldf = nxdata
 integer(4) :: i, j
 real(4) :: f, fdata(Ldf, nydata), pi, q, x, xdata(nxdata), y, ydata(nydata)
 logical(4) :: check = .true.                ! Выполняем проверку данных
 f(x, y) = sin(x + y)                        ! Задаем функцию
 do i = 1, nxdata                            ! Задаем сетку по x и y
  xdata(i) = float(i - 1) / float(nxdata - 1)
 end do
 do i = 1, nydata
  ydata(i) = float(i - 1) / float(nydata - 1)
 end do
 do i = 1, nxdata                            ! Задаем функцию на сетке
  do j = 1, nydata
   fdata(i, j) = f(xdata(i), ydata(j))
  end do
 end do                                      ! Вывод заголовка
 write(*, "(10x, 'x', 11x, 'y', 7x, 'f(x, y)', 6x, 'qd2vl', 9x, 'dif')")
 ! Получаем π и устанавливаем x и y
 pi = const('pi'); x = pi / 4.0; y = pi / 4.0
 ! Получаем и оцениваем квадратичный сплайн в точке (x, y)
 q = qd2vl(x, y, nxdata, xdata, nydata, ydata, fdata, nxdata, check)
 write(*, '(5f12.4)') x, y, f(x, y), q, (q - f(x, y))
end program qd2vlTest
```

Результат:

x	y	f(x, y)	qd2vl	dif
0.7854	0.7854	1.0000	1.0000	0.0000

1.6.5. ФУНКЦИЯ QD2DR (DQD2DR)

Оценивает в заданной точке (x, y) частную производную функции, определенной на прямоугольной двумерной сетке, используя квадратичную интерполяцию. Имеет вызов

result = QD2DR(*ixder, iyder, x, y, nxdata, xdata, nydata,* &
ydata, fdata, Ldf, check)

Все параметры функции QD2DR являются *входными*.

Результирующая переменная: QD2DR.

Описание параметров см. в табл. 1.5.

Комментарии:

1. Возможные информационные ошибки приведены в предшествующем разделе.
2. Поскольку интерполяция выполняется квадратичным многочленом, то при *ixder* > 2 и *iyder* > 2 функция QD2DR вернет нуль.

Описание функции QD2DR аналогично описанию функции QD2VL.

Пример. По таблице, содержащей 21×42 точек в прямоугольнике [0, 2]×[0, 2], выполняется квадратичная аппроксимация функции $\sin(x + y)$ и оценка сплайна и его производных в точке $x = y = \pi/3$.

```
program qd2drTest
 use dfimsl
 real(4) :: func
 ! Объявление иных данных см. в примере для QD2VL
 . . .
 ! Подготовку данных см. в примере для QD2VL
 . . .
 write (*, 1)                          ! Вывод заголовка таблицы результатов
 check = .true.                        ! Будет выполняться проверка данных
 ! Получаем π и устанавливаем x и y
 pi = const('pi'); x = pi / 3.0; y = pi / 3.0
 do ixder = 0, 1                       ! Оцениваем сплайн и его производные
  do iyder = 0, 1
   q = qd2dr(ixder, iyder, x, y, nxdata, xdata, nydata, ydata, fdata, Ldf, check)
   fu = func(ixder, iyder, x, y)
   write(*, "(2f9.4, 2i5, 3x, f9.4, 2x, 2f11.4)") x, y, ixder, iyder, fu, q, (fu - q)
  end do
 end do
 1 format(32x, '(idx, idy)', /, 8x, 'x', 8x, 'y', 3x, 'idx', 2x, 'idy', 3x, 'f(x, y)', 3x, 'qd2dr', 6x, 'Error')
end program qd2drTest

real function func(ix, iy, x, y)
 integer ix, iy
 real x, y
 if(ix == 0 .and. iy == 0) then        ! Задаем функцию
  func = sin(x + y)
 else if(ix == 0 .and. iy ==1) then    ! Задаем частную производную (0, 1)
  func = cos(x + y)
 else if(ix == 1 .and. iy == 0) then   ! Задаем частную производную (1, 0)
```

```
 func = cos(x+y)
else if(ix == 1 .and. iy == 1) then      ! Задаем частную производную (1, 1)
 func = -sin(x+y)
else
 func = 0.0
end if
end function func
```

Результат:

		(*idx*, *idy*)				
x	*y*	*idx*	*idy*	f(*x*, *y*)	q*d2dr*	*error*
1.0472	1.0472	0	0	0.8660	0.8658	-0.0002
1.0472	1.0472	0	1	-0.5000	-0.5043	-0.0043
1.0472	1.0472	1	0	-0.5000	-0.5045	-0.0045
1.0472	1.0472	1	1	-0.8660	-0.9240	-0.0580

1.6.6. ФУНКЦИЯ QD3VL (DQD3VL)

Оценивает в заданной точке (*x*, *y*, *z*) функцию, определенную на прямоугольной трехмерной сетке, используя квадратичную интерполяцию. Имеет вызов

result = QD3VL(*x*, *y*, *z*, *nxdata*, *xdata*, *nydata*, *ydata*, & *nzdata*, *zdata*, *fdata*, *Ldf*, *mdf*, *check*)

Все параметры функции QD3VL являются *входными*.

Результирующая переменная: QD3VL.

Описание параметров см. в табл. 1.5.

Комментарий. При работе с QD3VL могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
4	9	Вектор *xdata* не является строго возрастающим
4	10	Вектор *ydata* не является строго возрастающим
4	11	Вектор *zdata* не является строго возрастающим

Описание:

Функция QD3VL выполняет квадратичную интерполяцию функции, заданную таблицей данных (x_i, y_j, z_k, f_{ijk}) для $i = 1, \ldots, n_x$, $j = 1, \ldots, n_y$ и $k = 1, \ldots, n_z$, и возвращает значение квадратичного сплайна в заданной точке. Квадратичный сплайн q строится по 10 точкам. Первые семь - это точки (x_i, y_j, z_k), $(x_{i\pm1}, y_j, z_k)$, $(x_i, y_{j\pm1}, z_k)$ и $(x_i, y_j, z_{k\pm1})$. Остальные три берутся из октанта, содержащего (x, y, z). Функция QD3VL вернет значение $q(x, y, z)$.

1.6.7. ФУНКЦИЯ QD3DR (DQD3DR)

Оценивает в заданной точке (*x, y, z*) частную производную функции, определенной на прямоугольной трехмерной сетке, используя квадратичную интерполяцию. Имеет вызов

result = QD3DR(*ixder, iyder, izder, x, y, z, nxdata, xdata,* &
nydata, ydata, nzdata, zdata, fdata, Ldf, mdf, check)

Все параметры функции QD3DR являются *входными.*

Результирующая переменная: QD3DR.

Описание параметров см. в табл. 1.5.

Комментарий. При работе с QD3DR могут возникать те же, что и для функции QD3VL, информационные ошибки.

Описание функции QD3DR аналогично описанию функции QD3VL.

1.7. ИНТЕРПОЛЯЦИЯ ПО РАССЕЯННЫМ ДАННЫМ. ПОДПРОГРАММА SURF (DSURF)

Находит по произвольно распределенным данным двумерный интерполяционный сплайн пятой степени. Имеет вызов

CALL SURF(*ndata, xydata, fdata, nxout, nyout, xout, yout, sur, Ldsur*)

Параметры подпрограммы SURF:

Параметр *sur* является *выходным*, остальные параметры - *входные.*

ndata - число точек; не может быть менее четырех.

xydata - массив формы (2, *ndata*), содержащий координаты точек интерполяции. Причем точки не должны быть совпадающими; *x*-координата *i*-й точки заносится в *xydata*(1, *i*), а ее *y*-координата - в *xydata*(2, *i*).

fdata - вектор размера *ndata*, содержащий значения функции в точках интерполяции. Причем *fdata*(*i*) - значение функции в точке (*xydata*(1, *i*), *xydata*(2, *i*)).

nxout (*nyout*) - число элементов в векторе *xout* (*yout*).

xout (*yout*) - вектор размера *nxout* (*nyout*), содержащий возрастающую последовательность точек - *x*-координат (*y*-координат) сетки, на которой будет оцениваться интерполяционная сплайновая поверхность.

sur - массив формы (*Ldsur*, *nyout*). Содержит *nxout*×*nyout* значений найденного интерполяционного сплайна в узлах сетки (*xout*×*yout*), т. е. *sur*(*i, j*) - значение сплайна в точке (*xout*(*i*), *yout*(*j*)).

Ldsur - ведущий размер массива *sur*; не должен быть меньше *nxout*.

Комментарии:

1. При работе с SURF могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
4	5	Точки *xydata* не должны быть совпадающими
4	6	Вектор *xout* должен быть строго возрастающим
4	7	Вектор *yout* должен быть строго возрастающим

2. Используемый в SURF метод интерполяции позволяет воспроизводить линейные функции.

Описание:

Подпрограмма предназначена для вычислений C^1-непрерывной КМ-поверхности по рассеянным данным $\{(x_i, y_i, f_i)\}_{i=1}^n \in \mathbf{R}^3$. В результате оценки этой поверхности на заданной пользователем сетке подпрограмма SURF возвращает значения интерполяционного КМ-сплайна *s*. Порядок вычисления *s* таков:

1) выполняется триангуляция Делоне на наборе точек $\{(x_i, y_i)\}_{i=1}^n$;

2) в каждом полученном треугольнике T кусок поверхности имеет вид

$$s(x, y) = \sum_{m+n\le 5} c_{mn}^T x^m y^n \quad \forall x, y \in T$$

и вдобавок $s(x_i, y_i) = fi$ для $i = 1, \ldots, n$ и *s* является непрерывно дифференцируемой на границах соседних треугольников. Подробно метод изложен в [13].

Пример. По 20 расположенным на окружности радиуса 3 точкам вычисляется интерполяционный КМ-сплайн для линейной функции $3 + 7x + 2y$. Результат выводится на сетке размера 3×3 и в виде графика, создаваемого ОМ на сетке размера 9×9.

```
program surfTest
 use dfimsl
 integer(4), parameter :: ndata = 20, nxout = 3, nyout = 3, Ldsur = nxout, nsx = 9, nsy = 9
 integer(4) :: i, j, kb
 real(4) :: f, fdata(ndata), pi, x, xydata(2, ndata), y
 ! spline - массив, введенный для графического                                    &
 ! отображения КМ-сплайна средствами ОМ
 real(4), allocatable :: sur(:, :), xout(:), yout(:), spline(:, :)
 !dec$attributes array_visualizer :: spline
 f(x, y) = 3.0 + 7.0 * x + 2.0 * y          ! Задаем функцию
 pi = const('pi')                           ! Получаем число π
 do i = 1, ndata                            ! Задаем x, y и fdata на окружности
  xydata(1, i) = 3.0 * sin(2.0 * pi * float(i - 1) / float(ndata))
  xydata(2, i) = 3.0 * cos(2.0 * pi * float(i - 1) / float(ndata))
```

```
 fdata(i) = f(xydata(1, i), xydata(2, i))
end do
! Задаем векторы xout и yout на сетке [0,1]×[0,1] размера nxout×nyout
allocate(sur(Ldsur, nyout), xout(nxout), yout(nyout))
do i =1, nxout
  xout(i) = float(i - 1) / float(nxout - 1)
end do
do i =1, nyout
  yout(i) = float(i - 1) / float(nyout - 1)
end do
! Выполняем интерполяцию и возвращаем результат на сетке размера 3×3
call surf(ndata, xydata, fdata, nxout, nyout, xout, yout, sur, Ldsur)
! Вывод заголовка и результатов
write(*, "(' ', 10x, 'x', 11x, 'y', 9x, 'surf', 6x, 'f(x, y)', 6x, 'Error', /)")
do i = 1, nyout
 do j = 1, nxout
  write(*, 99999) xout(j), yout(i), sur(j, i), f(xout(j), yout(i)), abs(sur(j, i) - f(xout(j), yout(i)))
 end do
end do
99999 format (1x, 5f12.4)
! Задаем векторы xout и yout на сетке [0,1]×[0,1] размера nsx×nsy
deallocate(sur, xout, yout)
allocate(sur(nsx, nsy), xout(nsx), yout(nsy), spline(3, nsx * nsy))
do i =1, nsx
  xout(i) = float(i - 1) / float(nsx - 1)
end do
do i =1, nsy
  yout(i) = float(i - 1) / float(nsy - 1)
end do
! Выполняем интерполяцию и возвращаем результат на сетке размера 3×3
call surf(ndata, xydata, fdata, nsx, nsy, xout, yout, sur, nsx)
! Графическое отображение результата приведено на рис. 1.7
kb = 1
do i = 1, nsx
 spline(1, kb:kb + nsy - 1) = xout(i)
 spline(2, kb:kb + nsy - 1) = yout(1:nsy)
 spline(3, kb:kb + nsy - 1) = sur(i, 1:nsy)
 kb = kb + nsy
end do                                  ! Текст подпрограммы vGraph2 см. выше
call vGraph2(spline, nsx * nsy)         ! Вызов ОМ
deallocate(sur, xout, yout, spline)
end program surfTest
```

Результат:

x	y	surf	f(x, y)	Error
0.0000	0.0000	3.0000	3.0000	0.0000
0.5000	0.0000	6.5000	6.5000	0.0000
1.0000	0.0000	10.0000	10.0000	0.0000
0.0000	0.5000	4.0000	4.0000	0.0000
0.5000	0.5000	7.5000	7.5000	0.0000
1.0000	0.5000	11.0000	11.0000	0.0000
0.0000	1.0000	5.0000	5.0000	0.0000
0.5000	1.0000	8.5000	8.5000	0.0000
1.0000	1.0000	12.0000	12.0000	0.0000

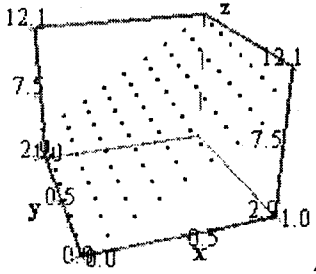


Рис. 1.7. КМ-сплайн для линейной функции 3 + 7x + 2y

1.8. АППРОКСИМАЦИЯ ПО МЕТОДУ НАИМЕНЬШИХ КВАДРАТОВ

1.8.1. ПЕРЕЧЕНЬ И ПАРАМЕТРЫ ПРОЦЕДУР

Процедуры, сглаживающие данные по методу наименьших квадратов, приведены в табл. 1.8, их повторяющиеся параметры - в табл. 1.9.

Таблица 1.8. Процедуры, формирующие сплайны по методу наименьших квадратов

Процедуры	*Описание*
RLINE	Находит регрессию, т. е. зависимость среднего значения одной величины от некоторой другой величины или нескольких величин, используя многочлен первой степени
RCURV	Находит регрессию, используя многочлены произвольной степени
FNLSQ	Находит регрессию, используя пользовательские базисные функции
BSLSQ и BSVLS	Выполняют приближение сплайнами соответственно по фиксированным и свободным узлам

CONFT	Возвращает по фиксированным узлам сплайн, вычисляемый по взвешенному методу наименьших квадратов с линейными ограничениям. Рекомендуется для употребления в тех случаях, когда важно сохранить форму кривой
BSLS2 и BSLS3	Вычисляют соответственно двумерные и трехмерные ТП-В-сплайновые регрессии

Таблица 1.9. Параметры процедур из табл. 1.8

Имя	*Смысл*	*Тип*
ndeg	Степень сглаживающего многочлена	INTEGER(4)
nobs	Число наблюдений	"
ssq	Квадратный корень из суммы квадратов отклонений (среднеквадратичная ошибка)	REAL(4) или REAL(8)
xguess	Вектор размера *ncoef* + *korder*, содержащий начальные координаты узлов; должен быть неубывающим	То же
xdata (*ydata*)	Вектор размера *nobs*, содержащий *x*-значения (*y*-значения) наблюдений	"
xweigh (*yweigh*, *zweigh*)	Вектор размера *nxdata* (*nydata*, *nzdata*), содержащий положительные веса элементов *xdata* (*ydata*, *zdata*)	"
weight	Вектор размера *ndata*, содержащий положительные веса точек	"

1.8.2. ОБОЗНАЧЕНИЯ В ФОРМУЛАХ

К обозначениям, введенным в разд. 1.4.1, добавляются следующие:

t^g - начальное предположение о последовательности узлов *xguess*;

w - вектор весов *weight*;

w_x (w_y, w_z) - вектор весов *xweigh* (*yweigh*, *zweigh*).

1.8.3. ПОДПРОГРАММА RLINE (DRLINE)

Сглаживает прямой линией $y = b_0 + b_1x$ по методу наименьших квадратов набор точек на плоскости. Имеет вызов

CALL RLINE(*nobs, xdata, ydata, b0, b1, stat*)

Параметры подпрограммы RLINE:

Входные: nobs, xdata, ydata.

Выходные: b0, b1, stat.

Смысл входных данных см. в табл. 1.9.

*b*0, *b*1 - коэффициенты результирующей регрессии $y = b_0 + b_1x$.

stat - вектор размера 12, содержащий следующие данные:

i	*stat*(*i*)
1	Среднее значение *xdata*
2	Среднее значение *ydata*
3	Дисперсия *xdata*
4	Дисперсия *ydata*
5	Корреляция
6	Ошибка оценки *b0*
7	Ошибка оценки *b1*
8	Степени свободы регрессии
9	Сумма квадратов отклонений для регрессии
10	Степени свободы ошибки
11	Сумма квадратов отклонений для ошибки
12	Число (*x*, *y*)-точек, содержащих NaN (не число) в *x*- или *y*-значении

Комментарий. При работе может возникнуть информационная ошибка типа 4 с кодом 1, означающая, что каждая (*x*, *y*)-точка содержит NaN.

Описание:

Метод, используемый в RLINE, обсуждается в [23]. Результирующая модель имеет вид $y = b_0 + b_1 x$.

Оценка стандартных ошибок *b*0 и *b*1 выполняется с применением простой линейной модели регрессии, в которой предполагается, что ошибки некоррелированны и имеют постоянную дисперсию.

Если все *x*-значения равны, то RLINE возвращает *b*1 = 0 и устанавливает в *b*0 среднее значение *y*-координат, хранящихся в *ydata*.

Пример. Подпрограмма RLINE аппроксимирует прямой данные, обсуждаемые в [23, табл. 1.1, с. 9-33)]. Отклик *y* - это количество используемого в месяц пара (в фунтах), а независимая переменная *x* - средняя атмосферная температура в градусах Фаренгейта.

```
program rlineTest
 use dfimsl
 integer(4), parameter :: nobs = 25
 real b0, b1, stat(12), xdata(nobs), ydata(nobs)
 character clabel(13)*15, rlabel(1)*4
 data xdata / 35.3, 29.7, 30.8, 58.8, 61.4, 71.3, 74.4, 76.7, 70.7, 57.5, 46.4, 28.9,      &
      28.1, 39.1, 46.8, 48.5, 59.3, 70.0, 70.0, 74.5, 72.1, 58.1, 44.6, 33.4, 28.6 /
 data ydata / 10.98, 11.13, 12.51, 8.4, 9.27, 8.73, 6.36, 8.5, 7.82, 9.14, 8.24, 12.19,   &
      11.88, 9.57, 10.94, 9.58, 10.09,8.11, 6.83, 8.88, 7.68, 8.47, 8.86, 10.36, 11.08 /
```

```
data rlabel / 'none' /, clabel / ' ', 'Mean of x', 'Mean of y', 'Variance x',          &
      'Variance y', 'Corr.', 'Std. Err. b0', 'Std. Err. b1', 'DF Reg.',                &
      'SS Reg.', 'DF Error', 'SS Error', 'Pts. with NaN' /
! Выполняем сглаживание
call rline(nobs, xdata, ydata, b0, b1, stat)
write(*, "(' b0 = ', f7.2, ', b1 = ', f9.5)") b0, b1
call wrrrl('%/stat', 1, 12, stat, 1, 0, '(12w10.4)', rlabel, clabel)
end program rlineTest                    ! Результат сглаживания см. также на рис. 1.8
```

Результат:

b0 = 13.62, b1 = -0.07983

stat

Mean of x	Mean of y	Variance x	Variance y	Corr.	Std. Err. b0
52.6	9.424	298.1	2.659	-0.8452	0.5815
Std. Err. b1	**DF Reg.**	**SS Reg.**	**DF Error**	**SS Error**	**Pts. with NaN**
0.01052	1	45.59	23	18.22	0

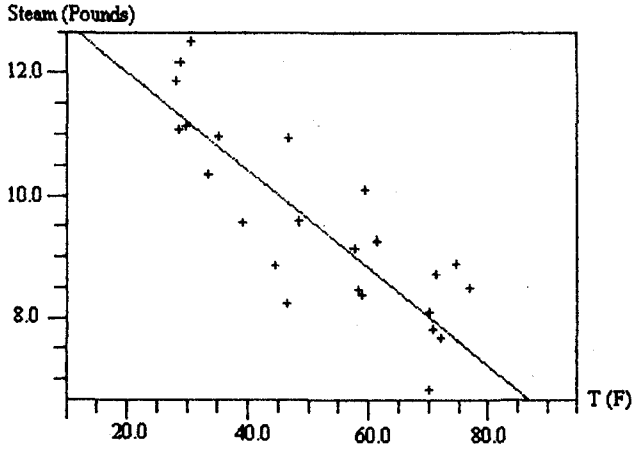


Рис. 1.8. Данные и аппроксимирующая линия

1.8.4. ПОДПРОГРАММА RCURV (DRCURV)

Сглаживает многочленом по методу наименьших квадратов набор точек на плоскости. Имеет вызов

CALL RCURV(*nobs, xdata, ydata, ndeg, b, sspoly, stat*)

Параметры подпрограммы RCURV:
Входные: nobs, xdata, ydata, ndeg.
Выходные: b, sspoly, stat.

Смысл входных данных см. в табл. 1.9.

b - вектор размера *ndeg* + 1, содержащий коэффициенты сглаживающего многочлена (*k* = *ndeg*)

$$y = b_0 + b_1x + b_2x^2 + \ldots + b_kx^k.$$

sspoly - вектор размера *ndeg* + 1, имеющий следующие значения:

- *sspoly*(*i*) содержит последовательные суммы квадратов отклонений для $i = 1, 2, \ldots, ndeg$;
- *sspoly*(*ndeg* + 1) содержит сумму квадратов отклонений x^i, вычисленную для средних значений $x, x_1, \ldots, x_i$.

stat - вектор размера 10, содержащий следующие данные:

i	*stat*(*i*)
1	Среднее значение *xdata*
2	Среднее значение *ydata*
3	Дисперсия *xdata*
4	Дисперсия *ydata*
5	Статистика R^2 (в процентах)
6	Степени свободы регрессии
7	Сумма квадратов отклонений для регрессии
8	Степени свободы ошибки
9	Сумма квадратов отклонений для ошибки
12	Число (*x*, *y*)-точек, содержащих NaN (не число) в *x*- или *y*-значении

Комментарии:

1. При работе с RCURV могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
4	3	Каждая (*x*, *y*)-точка содержит NaN
4	7	По крайней мере *ndeg* + 1 различных значений *x* необходимы для получения сглаживающего многочлена степени *ndeg*
3	4	Значения *y* постоянны. Сглаживающий многочлен имеет порядок 0
3	5	Слишком мало наблюдений, чтобы получить многочлен заданной степени. Коэффициенты при высоких степенях равны нулю
3	6	Хорошее сглаживание осуществляется многочленом, порядок которого меньше *ndeg*. Коэффициенты при высоких степенях равны нулю

2. При *ndeg* > 10 точность результата может быть неудовлетворительной.

Описание:

Последовательные суммы квадратов отклонений для каждой степени, возвращаемые в *sspoly*, полезны для понимания важности элементов сгла-

живающего многочлена с высокими степенями. Интерпретация этих сумм обсуждается в [23, с. 101-102] и [41, с. 278-287].

Статистика R^2, возвращаемая в *stat*(5), вычисляется по формуле

$$R^2 = \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} 100\%,$$

где $\hat{y}_i$ - сглаженное (среднее) значение y для x_i и $\bar{y}$, хранящегося в *stat*(2).

Эта статистика полезна для оценки качества сглаживания. Значение R^2 = 100 % говорит о совершенном сглаживании данных.

Подпрограмма RCURV использует ортогональные многочлены в качестве независимых переменных в уравнении регрессии, что позволяет повысить точность сглаживания. Подпрограмма RCURV основана на модификации алгоритма [25], приведенной в [50]. Модификация обсуждается в [37, с. 342-347].

Пример. Набор данных [41, с. 279-285], подлежащий сглаживанию, содержит отклики переменной *y*, измеряющей объем продаж кофе (в сотнях галлонов) от числа автоматов по разливу кофе. Отклики собраны по 14 кафетериям.

```
program rcurvTest
 use dfimsl
 integer(4), parameter :: ndeg = 2, nobs = 14, nval = 50
 integer(4) :: i
real(4) :: b(ndeg + 1), sspoly(ndeg + 1), stat(10), xdata(nobs), ydata(nobs), dx, xs, xe, fx
character clabel(11)*15, rlabel(1)*4
! Массив xyvalues введен для вывода графика сглаживающей кривой
real(kind(1e0)), allocatable :: xyvalues(:, :)
!dec$attributes array_visualizer :: xyvalues
data rlabel / 'none' /, clabel / ' ', 'Mean of x', 'Mean of y', 'Variance x',          &
      'Variance y', 'R-squared', 'DF Reg.', 'SS Reg.', 'DF Error',                     &
      'SS Error', 'Pts. with NaN' /
data xdata / 0.0, 0.0, 1.0, 1.0, 2.0, 2.0, 4.0, 4.0, 5.0, 5.0, 6.0, 6.0, 7.0, 7.0 /
data ydata / 508.1, 498.4, 568.2, 577.3, 651.7, 657.0, 755.3, 758.9,               &
      787.6, 792.1, 841.4, 831.8, 854.7, 871.4 /
call rcurv(nobs, xdata, ydata, ndeg, b, sspoly, stat)
call wrrrn('b', 1, ndeg+1, b, 1, 0)
call wrrrn('sspoly', 1, ndeg+1, sspoly, 1, 0)
call wrrrl('%/stat', 1, 10, stat, 1, 0, '(2w10.4)', rlabel, clabel)
! Вывод графика сглаживающей кривой; используем для вывода nval = 50 точек
allocate(xyvalues(2, nval))
```

```
 xs = xdata(1); xe = xdata(nobs)
 dx = (xe - xs) / float(nval - 1)
 do i = 1, nval
   xyvalues(1, i) = xs
   xyvalues(2, i) = fx(xs, b, ndeg)
  xs = xs + dx
 end do
 call vGraph(xyvalues, nval)          ! Результат см. на рис. 1.9 при ndata = 45
 deallocate(xyvalues)
end program rcurvTest

function fx(x, coeff, ndeg)           ! Оценивает сглаживающую кривую в точке x
 integer(4) :: ndeg, i                ! Вычисления выполняются по схеме Горнера
 real(4) :: fx, x, coeff(ndeg + 1)    ! Описание схемы см., например, в [5, с. 12]
 fx = coeff(ndeg + 1)
 do i = ndeg, 1, -1
  fx = coeff(i) + x * fx
 end do
end function fx
```

Результат:

b

503.3	78.9	-4.0

sspoly

7077152.0	220644.2	4387.7

stat

Mean of x	Mean of y	Variance x	Variance y	R-squared	DF Reg.
3.571	711.0	6.418	17364.8	99.69	2
SS Reg.	DF Error	SS Error	Pts. with NaN		
225031.9	11	711.2	0		

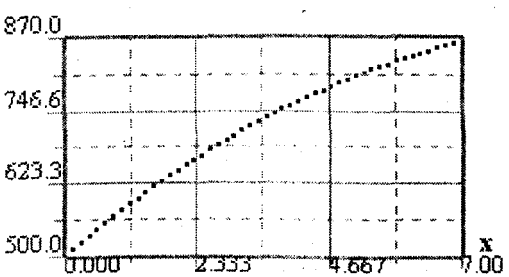


Рис. 1.9. Аппроксимирующая кривая

1.8.5. ПОДПРОГРАММА FNLSQ (DFNLSQ)

Выполняет по методу наименьших квадратов аппроксимацию, применяя пользовательские базисные функции. Имеет вызов

CALL FNLSQ(*f, intcep, nbasis, ndata, xdata, fdata, iwt, weight, a, sse*)

Параметры подпрограммы FNLSQ:

Пользовательская функция: f.

Входные: intcep, nbasis, ndata, xdata, fdata, iwt, weight.

Выходные: a, sse.

f - пользовательская функция, оценивающая базисные функции. Имеет вид *f*(*k, x*), где *k* – число базисных функций; *k* может равняться 1, 2, ..., *nbasis*; *x* - точка, в которой оценивается *k*-я базисная функция. Параметр *f* должен в вызывающей программной единице получить атрибут EXTERNAL. Вектор *fdata* приближается функцией

$$a(1) * f(1, x) + a(2) * f(2, x) + \ldots + a(nbasis) * f(nbasis, x),$$

если *intcep* = 0, и функцией

$$a(1) + a(2) * f(1, x) + \ldots + a(nbasis + 1) * f(nbasis, x),$$

если *intcep* = 1.

intcep - параметр отсечения; принимает значения 0 и 1; смысл параметра см. выше.

nbasis - число базисных функций.

ndata - число точек, по которым выполняется аппроксимация.

xdata - вектор размера *ndata*, содержащий абсциссы точек.

fdata - вектор размера *ndata*, содержащий ординаты точек.

iwt - параметр веса; оказывает следующее действие:

- *iwt* = 0 - вес принимается равным единице;
- *iwt* = 1 - вес принимается равным *weight*.

weight - вектор размера *ndata*, содержащий неотрицательные веса точек. Если *iwt* = 0, вектор *weight* не адресуется и может иметь размер 1.

a - вектор размера *intcep* + *nbasis*, содержащий коэффициенты аппроксимирующей функции. Если *intcep* = 1, то *a*(1) содержит отсечение; элемент *a*(*intcep* + *i*) содержит коэффициенты *i*-й базисной функции.

sse - ошибка, равная сумме квадратов отклонений.

Комментарий. При работе с FNLSQ могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	16	Существует линейная зависимость между базисными функциями. Один или более элементов вектора *a* приравнены нулю
3	2	Существует линейная зависимость между постоянной функцией (*a*(1), когда *intcep* = 1) и базисными функциями. Один или более элементов вектора *a* приравнены нулю
4	1	Обнаружен отрицательный вес

Описание:

Подпрограмма FNLSQ вычисляет методом наименьших квадратов аппроксимацию по набору данных $\{(x_i, f_i)\}_{i=1}^{n}$, применяя m = *nbasis* базисных функций $\{F_j\}_{j=1}^{m}$.

Если *intcep* = 0, подпрограмма возвращает в *sse* ошибку, равную сумме квадратов отклонений, и вектор коэффициентов *a*, доставляющий минимум этой сумме:

$$\sum_{i=1}^{n} w_i \left(f_i - \sum_{j=1}^{m} a_j F_j(x_i) \right)^2 .$$

Если *intcep* = 1, вектор коэффициентов *a* минимизирует следующую, возвращаемую *sse* сумму квадратов отклонений:

$$\sum_{i=1}^{n} w_i \left(f_i - a_1 - \sum_{j=1}^{m} a_j F_j(x_i) \right)^2 .$$

Пример. Выполняется аппроксимация двух различающихся на δε функций:

$$1 + \sin x + 7\sin 3x + \delta\varepsilon,$$

где ε - случайное число из диапазона [-1, 1], δ = 0 для первой функции и δ = 1 - для второй. Функции оцениваются в 90 равномерно размещенных точках на отрезке [0, 6]. При аппроксимации применяются 4 базисные функции $\sin kx$, $k = 1, \ldots, 4$ с отсечением и без него.

```
program fnlsqTest
 use dfimsl
 integer(4), parameter :: nbasis = 4, ndata = 90
```

```
integer(4) :: i, intcep, iwt
real(4) :: a(nbasis + 1), f, fdata(ndata), g, rnoise, sse, weight(ndata), x, xdata(ndata)
external f
g(x) = 1.0 + sin(x) + 7.0 * sin(3.0 * x)
call rnset(1234579)                    ! Затравка датчика случайных чисел
do i = 1, ndata                        ! Задаем данные
  xdata(i) = 6.0 * (float(i - 1) / float(ndata - 1))
  fdata(i) = g(xdata(i))
end do
intcep = 0; iwt = 0                    ! Вычисляем аппроксимацию с intcep = 0
call fnlsq(f, intcep, nbasis, ndata, xdata, fdata, iwt, weight, a, sse)
! Вывод заголовка и результата
write(*, 6)
write(*, 9) sse, (a(i), i =1, nbasis)
intcep = 1                             ! Вычисляем аппроксимацию с intcep = 1
call fnlsq(f, intcep, nbasis, ndata, xdata, fdata, iwt, weight, a, sse)
write(*, 8) sse, a(1), (a(i), i = 2, nbasis + 1)
do i = 1, ndata                        ! Добавляем шум
  rnoise = 2.0 * rnunf( ) - 1.0
  fdata(i) = fdata(i) + rnoise
end do
intcep = 0; iwt = 0
call fnlsq(f, intcep, nbasis, ndata, xdata, fdata, iwt, weight, a, sse)
write(*, 7)
write(*, 9) sse, (a(i), i = 1, nbasis)
intcep = 1                             ! Вычисляем аппроксимацию с intcep = 1
call fnlsq(f, intcep, nbasis, ndata, xdata, fdata, iwt, weight, a, sse)
write(*, 8) sse, a(1), (a(i), i = 2, nbasis + 1)
 6 format(/, ' Without error introduced we have:', /, ' SSE intercept coefficients ', /)
 7 format(/, ' With error introduced we have:', /, ' SSE intercept coefficients ', /)
 8 format(1x, f8.4, 5x, f9.4, 5x, 4f9.4, /)
 9 format(1x, f8.4, 14x, 5x, 4f9.4, /)
end program fnlsqTest

real function f(k, x)
 integer k
 real x
 f = sin(k * x)
end function f
```

Результат:

```
Without error introduced we have:
SSE intercept coefficients
```

89.8775		1.0101	0.0199	7.0291	0.0374
0.0000	1.0000	1.0000	0.0000	7.0000	0.0000

With error introduced we have:

SSE Intercept Coefficients

112.4662		0.9963	-0.0675	6.9825	0.0133
30.9831	0.9522	0.9867	-0.0864	6.9548	-0.0223

1.8.6. ПОДПРОГРАММА BSLSQ (DBSLSQ)

Вычисляет по методу наименьших квадратов аппроксимирующий B-сплайн и возвращает его коэффициенты. Имеет вызов

CALL BSLSQ(*ndata, xdata, fdata, weight, korder, xknot, ncoef, bscoef*)

Параметр *bscoef* является *выходным*. Остальные параметры - *входные*. Смысл параметров, кроме *weight*, размещенного в табл. 1.9, см. в табл. 1.4.

Комментарии:

1. При работе с BSLSQ могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
4	5	Число повторяемости узлов не может превышать порядок сплайна
4	6	Узлы должны быть упорядочены по неубыванию значений
4	7	Все веса должны быть больше нуля
4	8	Наименьший элемент вектора *xdata* должен быть не меньше узла *korder*
4	9	Наибольший элемент вектора *xdata* должен быть не больше узла *ncoef* + 1

2. B-сплайн можно оценить функцией BSVAL, а его производные - BSDER.

Описание:

Подпрограмма BSLSQ вычисляет взвешенную дискретную L_2 аппроксимацию по данному набору данных (x_i, f_i), $i = 1, \ldots, n$ ($n = ndata$). Иными словами, коэффициенты B-сплайна $b = bscoef$ таковы, что минимальна сумма

$$\sum_{i=1}^{n} \left| f_i - \sum_{j=1}^{m} b_j B_j(x_j) \right|^2 w_i .$$

Задача наименьших квадратов сводится к решению линейной системы нормальных уравнений. Вычислительные проблемы, иногда возникающие при решении нормальных уравнений, здесь не обнаруживаются, поскольку

B-сплайновый базис приводит в общем случае к хорошо обусловленной ленточной матрице.

Веса в некоторых случаях определяются относительной важностью входных данных. При аппроксимации непрерывных функций целесообразнее выбирать квадратурные веса и точки Гаусса. Это обусловлено тем, что BSLSQ минимизирует интеграл

$$\int |F - s|^2 dx .$$

Квадратурные веса и точки Гаусса можно получить, вызвав процедуру GQRUL библиотеки IMSL.

Подпрограмма BSLSQ основана на процедуре L2APPR, приведенной в [7].

Пример. Вычисляется аппроксимация многочлена второй степени в виде квадратичного сплайна с одним внутренним узлом по двум различным наборам точек. Первый набор получается в результате оценки исходного многочлена в 50 равномерно распределенных точках на интервале (0, 1) с последующим добавлением равномерно распределенного шума. Второй набор данных включает первый и дополнительно значения без шума в точках 0 и 1. Причем для этих точек задается вес, равный 10^5. В качестве результата в 11 точках выводятся оценки функции, аппроксимирующих сплайнов и абсолютные ошибки. Пример демонстрирует возможности хорошего приближения к некоторым точкам (в примере - начало и конец отрезка) в результате задания для них существенных весов.

```
program bslsqTest
 use dfimsl
 integer(4), parameter :: korder = 3, ncoef = 4
 integer(4) :: i, ndata
 real(4) :: bscof1(ncoef), bscof2(ncoef), f, fdata1(50), fdata2(52), rnoise, s1,          &
      s2, weight(52), x, xdata1(50), xdata2(52), xknot(korder + ncoef), xt, yt
 data weight / 52*1.0 /
 f(x) = 8.0 * x * (1.0 - x)                  ! Задаем функцию
 call rnset(12345679)                        ! Затравка датчика случайных чисел
 ndata = 50
 do i = 1, ncoef - korder + 2                ! Задаем внутренние узлы
  xknot(i + korder - 1) = float(i - 1) / float(ncoef - korder + 1)
 end do
 ! Формируем совмещенные узлы в начале и конце отрезка аппроксимации
 do i = 1, korder - 1
   xknot(i) = xknot(korder)
   xknot(i + ncoef + 1) = xknot(ncoef + 1)
```

```
end do
! Задаем точки и добавляем шум к значениям функции
do i = 1, ndata
  xdata1(i) = float(i) / 51.0
  rnoise = (rnunf( ) - 0.5)
  fdata1(i) = f(xdata1(i)) + rnoise
end do
! Вычисляем сплайн
call bslsq(ndata, xdata1, fdata1, weight, korder, xknot, ncoef, bscof1)
! Теперь получим сплайн с теми же данными,
! но с большими весами в конечных точках
ndata = 52
xdata2(2:51) = xdata1
fdata2(2:51) = fdata1
weight(1) = 1.0e5
xdata2(1) = 0.0
fdata2(1) = f(xdata2(1))
weight(ndata) = 1.0e5
xdata2(ndata) = 1.0
fdata2(ndata) = f(xdata2(ndata))
! Вычисляем сплайн
call bslsq(ndata, xdata2, fdata2, weight, korder, xknot, ncoef, bscof2)
! Вывод заголовка и результата в 11 точках
write(*, "(7x, 'x', 9x, 'f(x)', 6x, 's1(x)', 5x, 's2(x)', 7x, 'f(x) - s1(x)', 7x, 'f(x) - s2(x)')")
do i = 1, 11
 xt = float(i - 1) / 10.0; yt = f(xt)        ! s1, s2 - оценки сплайна
 s1 = bsval(xt, korder, xknot, ncoef, bscof1)
 s2 = bsval(xt, korder, xknot, ncoef, bscof2)
 write(*, 99999) xt, yt, s1, s2, (s1 - yt), (s2 - yt)
end do
99999 format (' ', 4f10.4, 4x, f10.4, 7x, f10.4)
end program bslsqTest
```

Результат:

x	f(x)	s1(x)	s2(x)	f(x) - s1(x)	f(x) - s2(x)
0.0000	0.0000	0.0515	0.0000	0.0515	0.0000
0.1000	0.7200	0.7594	0.7490	0.0394	0.0290
0.2000	1.2800	1.3142	1.3277	0.0342	0.0477
0.3000	1.6800	1.7158	1.7362	0.0358	0.0562
0.4000	1.9200	1.9641	1.9744	0.0441	0.0544
0.5000	2.0000	2.0593	2.0423	0.0593	0.0423
0.6000	1.9200	1.9842	1.9468	0.0642	0.0268

0.7000	1.6800	1.7220	1.6948	0.0420	0.0148
0.8000	1.2800	1.2726	1.2863	-0.0074	0.0063
0.9000	0.7200	0.6360	0.7214	-0.0840	0.0014
1.0000	0.0000	-0.1878	0.0000	-0.1878	0.0000

1.8.7. ПОДПРОГРАММА BSVLS (DBSVLS)

Вычисляет по методу наименьших квадратов аппроксимирующий B-сплайн с изменяющимися узлами и возвращает его коэффициенты. Имеет вызов

CALL BSVLS(*ndata, xdata, fdata, weight, korder, ncoef,* &
xguess, xknot, bscoef, ssq)

Параметры *bscoef* и *ssq* являются *выходными*. Остальные параметры - *входные*. Смысл параметров, кроме *weight*, *xguess* и *ssq*, размещенных в табл. 1.9, см. в табл. 1.4.

Комментарии:

1. При работе с BSVLS могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	12	Число оптимальных совмещенных узлов больше, чем *korder*. Это означает, что при меньшем числе узлов будет получено то же среднеквадратичное отклонение. Узлы будут слегка разъединены
4	9	Число повторяемости узлов в *xguess* не может превышать порядок сплайна
4	10	Вектор *xguess* должен быть неубывающим

2. B-сплайн можно оценить функцией BSVAL, а его производные - BSDER.

Описание:

Подпрограмма BSVLS пытается найти наилучшее расположение узлов, через которые проходит B-сплайн, позволяющее минимизировать ошибку метода наименьших квадратов. Подпрограмма имеет эффект, если $n_c > k$. В процессе вычислений минимизируется функционал

$$F(b,t) = \sum_{i=1}^{n} w_i \left(f_i - \sum_{j=1}^{n_c} b_j B_{jkt}(x_j) \right)^2 .$$

Минимум ищется по всем допустимым последовательностям узлов t.

Известно, что для фиксированной последовательности узлов t поиск оптимальных коэффициентов b является линейной задачей наименьших квад-

ратов, которая может быть решена процедурой BSLSQ. Таким образом, функционал F можно рассматривать как целевую функцию от t:

$$G(t) = \min_b F(b, t).$$

Снижение значения целевой функции G осуществляется методом Гаусса - Зейделя. Дополнительно в алгоритм включен эвристический поиск, который эффективен, когда данные порождаются гладкой функцией. Эвристика основывается на процедуре NEWNOT, приведенной в [7].

Начальное предположение t^g о последовательности узлов для сплайна порядка k должно содержать допустимую неубывающую последовательность координат, такую, что

$$t_1^g \le \ldots \le t_k^g \le x_i \le t_{n+1}^g \le \ldots \le t_{n+k}^g,\ i = 1, \ldots m$$

и

$$t_1^g < t_{i+k}^g,\ i = 1, \ldots n.$$

Подпрограмма BSVLS возвращает наилучшее найденное представление B-сплайна и в *ssq* квадратный корень из суммы квадратов отклонений. Если ответ неудовлетворителен, можно выполнить инициализацию BSVLS полученными данными и проверить, сможет ли BSVLS дать улучшение. Заметим, однако, что такой прием не приносит, как правило, существенных улучшений результата. Время работы подпрограммы BSVLS может быть на несколько порядков больше, чем один вызов рассмотренной выше подпрограммы BSLSQ.

Пример. Выполняется сглаживание функции $|x - 0.33|$ по 100 равномерно размещенным на отрезке [0, 1] точкам. Первоначально употребляется квадратичный сплайн с двумя внутренними узлами, имеющими начальные значения 0.2 и 0.8. Конечная (теоретическая) ошибка должна быть равна нулю, поскольку два узла квадратичного сплайна сгруппированы в точке 0.33. Затем с теми же данными выполняется сглаживание кубическим сплайном с тремя внутренними узлами с начальными значениями 0.1, 0.2 и 0.5. Вновь теоретическая ошибка равна нулю, когда 3 узла совмещены в точке 0.33. На рис. 1.10 приводятся графики B-сплайнов, полученных подпрограммой BSLSQ с узлами в 0.2 и 0.8 и подпрограммой BSVLS. Последний график неотличим от графика функции $|x - 0.33|$.

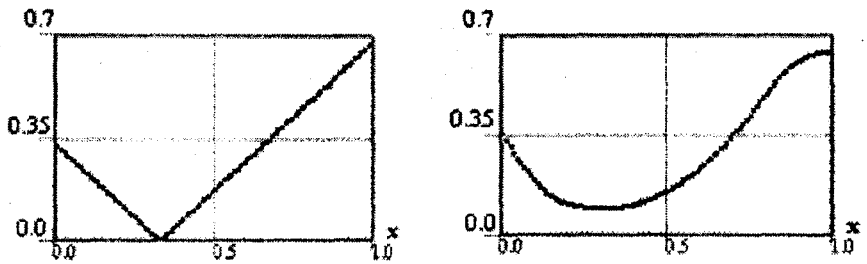
```
program bsvlsTest
 use dfimsl
 integer(4), parameter :: kord1 = 3, kord2 = 4, ncoef1 = 5, ncoef2 = 7, ndata = 100
 integer(4) :: i
```

```
real(4) :: bscoef(ncoef2), f, fdata(ndata), ssq, weight(ndata), x, xdata(ndata),          &
      xgues1(ncoef1 + kord1), xgues2(kord2 + ncoef2), xknot(ncoef2 + kord2)
data xgues1 / 3*0.0, 0.2, 0.8, 3*1.0001 /
data xgues2 / 4*0.0, 0.1, 0.2, 0.5, 4*1.0001 /
data weight / ndata*0.01 /
f(x) = abs(x - 0.33)                          ! Задаем функцию и точки
do i = 1, ndata
  xdata(i) = float(i - 1) / float(ndata - 1)
  fdata(i) = f(xdata(i))
end do
! Вычисляем представление B-сплайна с kord1, ncoef1 и xgues1
call bsvls(ndata, xdata, fdata, weight, kord1, ncoef1, xgues1, xknot, bscoef, ssq)
write(*, 1) 'Quadratic'
! Вывод ssq и узлов
write(*, 2) ssq, (xknot(i), i = 1, kord1 + ncoef1)
! Выводим график сплайна, вычисленного BSVLS с kord1, ncoef1 и xgues1
call plotSpline(ndata, 1.0, kord1, xknot(1:ncoef1 + kord1), ncoef1, bscoef, 0.0)
! Вычисляем представление B-сплайна с kord2, ncoef2 и xgues2
call bsvls(ndata, xdata, fdata, weight, kord2, ncoef2, xgues2, xknot, bscoef, ssq)
write(*, 1) 'Cubic'
write(*, 2) ssq, (xknot(i), i = 1, kord2 + ncoef2)
1 format(' Piecewise ', a, /)
2 format(' Square root of the sum of squares: ', f9.4, /, ' Knot sequence: ', /, 1x, 11(f9.4, 1x))
! Вычисляем теперь сплайн, применив BSLSQ
xknot = 0.0
xknot(1:ncoef1 + kord1) = xgues1
call bslsq(ndata, xdata, fdata, weight, kord1, xknot, ncoef1, bscoef)
! Выводим график сплайна, вычисленного BSLSQ
call plotSpline(ndata, 1.0, kord1, xknot(1:ncoef1 + kord1), ncoef1, bscoef, 0.0)
contains
! Вывод графика сплайна
! Используем режим векторного графа OM
subroutine plotSpline(ndata, grdsiz, korder, xknot, ncoef, bscoef, dif)
 integer(4) :: ndata, korder, ncoef, i
 real(4) :: grdsiz, xknot(ncoef + korder), bscoef(ncoef), dif
 ! Массив xyvalues введен для вывода графика сплайна
 real(kind(1e0)), allocatable :: xyvalues(:, :)
 !dec$attributes array_visualizer :: xyvalues
 allocate(xyvalues(2, ndata))
 do i = 1, ndata                              ! Выполняем оценку сплайна
  xyvalues(1, i) = grdsiz * float(i - 1) / float(ndata - 1) - dif
  xyvalues(2, i) = bsval(xyvalues(1, i), korder, xknot, ncoef, bscoef)
```

```
  end do
  call vGraph(xyvalues, ndata)          ! Результат см. на рис. 1.10, а
  deallocate(xyvalues)                  ! Текст vGraph см. выше
 end subroutine plotSpline
end program bsvlsTest
```

Результат:

```
Piecewise quadratic
Square root of the sum of squares : 0.0011
Knot sequence:
0.0000 0.0000 0.0000 0.3069 0.3506 1.0001 1.0001 1.0001
Piecewise cubic
Square root of the sum of squares: 0.0005
Knot sequence:
0.0000 0.0000 0.0000 0.0000 0.3167 0.3273 0.3464 1.0001 1.0001 1.0001 1.0001
```



а *б*

Рис. 1.10. Аппроксимация функции |x - 0.33|:
а - подпрограммой BSVLS; б - подпрограммой BSLSQ

1.8.8. ПОДПРОГРАММА CONFT (DCONFT)

Вычисляет по методу наименьших квадратов аппроксимирующий B-сплайн с ограничениями и возвращает его коэффициенты. Имеет вызов

```
CALL CONFT(ndata, xdata, fdata, weight, nxval, xval, nhard,          &
     ider, itype, bl, bu, korder, xknot, ncoef, bscoef)
```

Параметры подпрограммы CONFT:

Параметр *bscoef* является *выходным*. Остальные параметры - *входные*. Смысл параметров, кроме приведенных ниже, см. в табл. 1.4 и 1.9.

nxval - число точек в векторе *xval*.

xval - вектор размера *nxval*, содержащий абсциссы точек, в которых на сплайн накладываются ограничения.

nhard - число элементов в *xval*, имеющих твердые ограничения ($0 \le nhard \le nxval$). Задание $nhard = nxval$ приведет к выполнению всех ограничений. Твердые ограничения должны быть выполнены. Если это невозможно, подпрограмма сообщит о неудаче. Мягкие ограничения могут не выполняться. Однако подпрограмма всегда пытается их удовлетворить. Ограничения должны быть упорядочены по приоритету. На первом месте размещается наиболее важное. Таким образом, твердые ограничения предшествуют мягким. Мягкие ограничения удовлетворяются либо полностью, либо в максимально возможном объеме в порядке приоритета.

ider - вектор размера *nxval*, содержащий значения производных сплайна, которым он должен удовлетворять. Если необходимо ограничить интеграл сплайна на отрезке $[c, d]$, следует задать $ider(i) = ider(i + 1) = -1$ и $xval(i) = c$, $xval(i + 1) = d$. Необходимо, чтобы $itype(i) = itype(i + 1) \ge 0$ и $c \le d$.

itype - вектор размера *nxval*, задающий типы ограничений. Принимает следующие значения:

itype(*i*)	*i*-е ограничение	Примечание
1	$bl(i) = f^{(d_i)}(x_i)$	-
2	$f^{(d_i)}(x_i) = bu(i)$	-
3	$f^{(d_i)}(x_i) = bl(i)$	-
4	$bl(i) \le f^{(d_i)}(x_i) \le bu(i)$	-
1	$bl(i) = \int_c^d f^{(d_i)}(x_i)$	$d_i = -1$
2	$\int_c^d f^{(d_i)}(x_i) \le bu(i)$	$d_i = -1$
3	$\int_c^d f^{(d_i)}(x_i) \ge bl(i)$	$d_i = -1$
4	$bl(i) \le \int_c^d f^{(d_i)}(x_i) \le bu(i)$	$d_i = -1$
10	Периодические граничные условия	
99	Игнорировать это ограничение	

Чтобы задать двухточечное ограничение, необходимо иметь *itype*(*i*) = = *itype*(*i* + 1) и *itype*(*i*) должен быть отрицателен. Возможны следующие двухточечные ограничения:

itype(*i*)	*i*-е ограничение
-1	$bl(i) = f^{(d_i)}(x_i) - f^{(d_{i+1})}(x_{i+1})$
-2	$f^{(d_i)}(x_i) - f^{(d_{i+1})}(x_{i+1}) \le bu(i)$
-3	$f^{(d_i)}(x_i) - f^{(d_{i+1})}(x_{i+1}) \ge bl(i)$
-4	$bl(i) \le f^{(d_i)}(x_i) - f^{(d_{i+1})}(x_{i+1}) \le bu(i)$

bl (*bu*) - вектор размера *nxval*, содержащий нижнюю (верхнюю) границу ограничения; если *i*-е ограничение не имеет нижней (верхней) границы, то элемент *bl*(*i*) (*bu*(*i*)) не адресуется. Если нет ограничений диапазона, например с *itype*(*i*) = -1, то *bl* и *bu* могут адресовать одну память (быть одним массивом). Если *i*-е ограничение - это ограничение равенства, то *bu*(*i*) не адресуется.

Комментарий. При работе с CONFT могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	11	Чтобы получить приближение, мягкое ограничение будет удалено
4	12	Число повторяемости узлов не может превышать порядок сплайна
4	13	Узлы должны быть неубывающими
4	14	Наименьший элемент вектора *xdata* должен быть не меньше узла *korder*
4	15	Наибольший элемент вектора *xdata* должен быть не больше узла *ncoef* + 1
4	16	Все веса должны быть больше нуля
4	17	Нельзя удовлетворить твердые ограничения
4	18	Абсциссы ограничений должны размещаться в пределах интервала задания узлов
4	19	Верхняя граница диапазонного ограничения должна быть не меньше его нижней границы
4	20	При ограничении интеграла сплайна верхняя граница интегрирования должна быть не меньше нижней границы

Описание:

Подпрограмма CONFT вычисляет, используя метод наименьших квадратов, ограниченный, взвешенный сплайн. Ограничения подразделяются на одноточечные, двухточечные и интегралы. Поддерживаются следующие 4 типа ограничений:

$$E_p[f]=\begin{cases} f^{(j_p)}(y_p); \\ f^{(j_p)}(y_p)-f^{(j_{p+1})}(y_{p+1}); \\ \int\limits_{y_p}^{y_{p+1}} f(t); \\ \text{периодические граничные}. \end{cases}$$

Интервал I_p, который может быть точкой, конечным или полуконечным, ассоциируется с определенным ограничением. Алгоритм пытается удовлетворить все ограничения. Если же это невозможно, то он отбрасывает мягкие ограничения в обратном порядке их размещения в векторе *xval* до тех пор, пока он не найдет приемлемый набор ограничений или не обнаружит, что нельзя выполнить твердое ограничение, заданное элементами *xval*(1), ..., *xval*(*nhard*). Если ограничение отброшено, пользователь получит соответствующее уведомление.

Таким образом, подпрограмма CONFT решает задачу

$$\sum_{i=1}^{n}\left|f_i-\sum_{j=1}^{m}b_jB_j(x_i)\right|^2 w_i$$

с ограничениями

$$E_p\left[\sum_{j=1}^{m}b_jB_j(x_i)\right]\in I_p,\ p=1,\ldots,n_f,$$

где n_f - число выполнимых ограничений. Эта линейная задача наименьших квадратов рассматривается IMSL как задача квадратичного программирования и решается с помощью подпрограммы QPROG.

Выбор весов часто зависит от оценки достоверности тех или иных точек.

Определение выполнимых линейных ограничений - задача, чувствительная к входным данным. При возникновении трудностей один из способов их преодоления - увеличение интервала I_p задания ограничения.

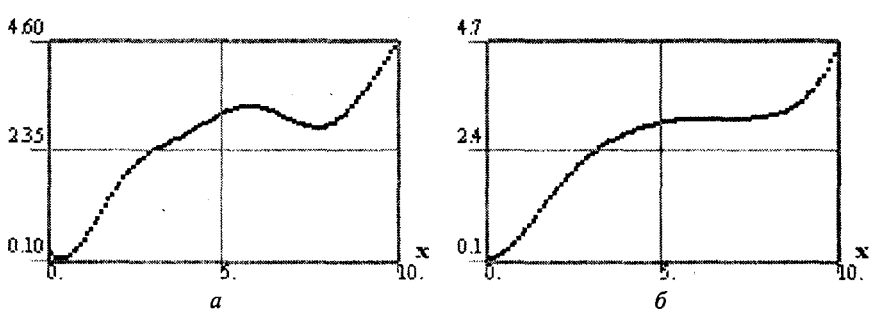
Пример 1. Выполняется сглаживание кубическим сплайном данных, порожденных функцией $f(x) = x/2 + \sin(x/2)$ и загрязненных случайным шумом. Функция на исследуемом отрезке является возрастающей, однако подпрограмма BSLSQ не обеспечивает адекватного сглаживания (что видно из рис. 1.11). Чтобы добиться хорошего сглаживания посредством CONFT, в *nxval* = 15 точках задается большая нуля производная. Поэтому результирующая кривая является монотонно возрастающей. Для двух приближений, выполненных подпрограммами BSLSQ и CONFT, выводятся ошибки на 100 равномерно размещенных точках и графики.

```
program conftTest1
 use dfimsl
 integer(4), parameter :: korder = 4, ncoef = 8, ndata = 15, nxval = 15
 integer(4) :: i, ider(nxval), itype(nxval), nhard
 real(4) :: bl(nxval), bsclsq(ndata), bscnft(ndata), bu(nxval), errlsq, errnft, f1,      &
      fdata(ndata), grdsiz, weight(ndata), x, xdata(ndata),                              &
      xknot(korder + ndata), xval(nxval)
 f1(x) = 0.5 * x + sin(0.5 * x)              ! Задаем функцию
 call rnset(234579)                          ! Затравка датчика случайных чисел
 weight = 1.0                                ! Вес точек равен 1.0
 grdsiz = 10.0                               ! Вычисляем исходные данные с шумом
 do i = 1, ndata                             ! и добавляем в них случайный шум
  xdata(i) = grdsiz * ((float(i - 1) / float(ndata - 1)))
  fdata(i) = f1(xdata(i)) + (rnunf( ) - 0.5)
 end do
 do i = 1, ncoef - korder + 2                ! Вычисляем узлы
  xknot(i + korder - 1) = grdsiz * ((float(i - 1) / float(ncoef - korder + 1)))
 end do
 do i = 1, korder - 1
   xknot(i) = xknot(korder)
   xknot(i + ncoef + 1) = xknot(ncoef + 1)
 end do
 ! Выполняем сглаживание подпрограммой BSLSQ
 call bslsq(ndata, xdata, fdata, weight, korder, xknot, ncoef, bsclsq)
 ! Выводим график сплайна, вычисленного BSLSQ
 call plotSpline(100, grdsiz, korder, xknot, ncoef, bsclsq, 0.0)
 do i = 1, nxval                             ! Задаем ограничения для CONFT
   xval(i) = grdsiz * float(i - 1) / float(nxval - 1)
   itype(i) = 3; ider(i) = 1; bl(i) = 0.0
 end do
 nhard = 0                                   ! Вызываем CONFT
 call conft(ndata, xdata, fdata, weight, nxval, xval, nhard,                            &
      ider, itype, bl, bu, korder, xknot, ncoef, bscnft)
```

```
errlsq = 0.0; errnft = 0.0                    ! Вычисляем средние ошибки по 100 точкам
do i = 1, 100
 x = grdsiz * float(i - 1) / 99.0
 errnft = errnft + abs(f1(x) - bsval(x, korder, xknot, ncoef, bscnft))
 errlsq = errlsq + abs(f1(x) - bsval(x, korder, xknot, ncoef, bsclsq))
end do                                        ! Вывод результата
! Выводим график сплайна, вычисленного CONFT
call plotSpline(100, grdsiz, korder, xknot, ncoef, bscnft, 0.0)
write(*, "(' Average error with BSLSQ fit: ', f8.5)") errlsq / 100.0
write(*, "(' Average error with CONFT fit: ', f8.5)") errnft / 100.0
contains
! Вывод графика сплайна
! Используем режим векторного графа OM
subroutine plotSpline(ndata, grdsiz, korder, xknot, ncoef, bscoef, dif)
! Текст plotSpline см. выше
...
end program conftTest1                        ! Графики см. на рис. 1.11
```

Результат:

```
Average error with BSLSQ fit: 0.20250
Average error with CONFT fit: 0.14334
```



а *б*

Рис. 1.11. Сглаживание зашумленных данных, порожденных функцией f(x) = x/2 + sin(x/2): а - подпрограммой BSLSQ; б - подпрограммой CONFT

Пример 2. Выполняется сглаживание кубическим сплайном данных, порожденных функцией $f(x) = 1/(1 + x^4)$ и загрязненных случайным шумом. Первоначально задача решается BSLSQ. Полученное сглаживание неудовлетворительно, поэтому задаются ограничения и вызывается CONFT. Задавая ограничения, прежде всего отметим, что решение, полученное посредством BSLSQ, колеблется около исходной функции в начале и конце интервала. Это обычное для одномерных данных явление. Чтобы устранить колебания, задаем для кубического сплайна нулевые вторые производные

в трех первых и трех последних узлах. Это приведет к тому, что на интервалах между этими узлами элементарные сплайны будут линейными. Дополнительно приближение s ограничиваем следующим образом: $s(-7) \geq 0$, $\int_{-7}^{7} s(x)dx \leq 2.3$ и $s(-7) = s(7)$. Заметьте, что последнее ограничение является периодическим, задаваемым для функции.

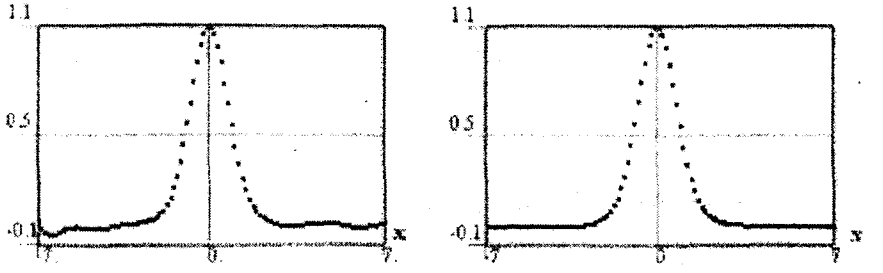
В качестве результата выводятся данные о средних ошибках по 100 точкам и графики двух приближений.

```
program conftTest2
 use dfimsl
 integer(4), parameter :: korder = 4, ncoef = 15, ndata = 51, nxval = 12
 integer(4) :: i, ider(nxval), itype(nxval), nhard
 real(4) :: bl(nxval), bsclsq(ndata), bscnft(ndata), bu(nxval), errlsq, errnft, f1,        &
       fdata(ndata), grdsiz, weight(ndata), x, xdata(ndata),                               &
       xknot(korder + ndata), xval(nxval)
 f1(x) = 1.0 / (1.0 + x**4)                    ! Задаем функцию
 call rnset(234579)                            ! Затравка датчика случайных чисел
 weight = 1.0                                  ! Вес точек равен 1.0
 grdsiz = 14.0                                 ! Вычисляем исходные данные с шумом
 do i = 1, ndata                               ! и добавляем в них случайный шум
  xdata(i) = grdsiz * ((float(i - 1) / float(ndata - 1))) - grdsiz / 2.0
  fdata(i) = f1(xdata(i)) + 0.125 * (rnunf( )-0.5)
 end do
 do i = 1, ncoef - korder + 2                  ! Вычисляем узлы
  xknot(i + korder - 1) = grdsiz * ((float(i - 1) / float(ncoef - korder + 1))) - grdsiz / 2.0
 end do
 do i = 1, korder - 1
   xknot(i) = xknot(korder)
   xknot(i + ncoef + 1) = xknot(ncoef + 1)
 end do
 ! Выполняем сглаживание подпрограммой BSLSQ
 call bslsq(ndata, xdata, fdata, weight, korder, xknot, ncoef, bsclsq)
 ! Выводим график сплайна, вычисленного BSLSQ
 call plotSpline(100, grdsiz, korder, xknot, ncoef, bsclsq, grdsiz / 2.0)
 do i = 1, 4                                   ! Задаем ограничения и вызываем CONFT
   xval(i) = xknot(korder + i - 1)
   xval(i + 4) = xknot(ncoef - 3 + i)
   itype(i) = 1; itype(i + 4) = 1
   ider(i) = 2; ider(i + 4) = 2
   bl(i) = 0.0; bl(i + 4) = 0.0
 end do
 xval(9) = -7.0; itype(9) = 3
```

```
 ider(9) = 0; bl(9) = 0.0
 xval(10) = -7.0; itype(10) = 2
 ider(10) = -1; bu(10) = 2.3
 xval(11) = 7.0; itype(11) = 2
 ider(11) = -1; bu(11) = 2.3
 xval(12) = -7.0; itype(12) = 10
 ider(12) = 0
 call conft(ndata, xdata, fdata, weight, nxval, xval, nharpt, ider, itype,                &
      bl, bu, korder, xknot, ncoef, bscnft)
 errlsq = 0.0; errnft = 0.0                  ! Вычисляем средние ошибки по 100 точкам
 do i = 1, 100
  x = grdsiz * float(i - 1) / 99.0 - grdsiz / 2.0
  errnft = errnft + abs(f1(x) - bsval(x, korder, xknot, ncoef, bscnft))
  errlsq = errlsq + abs(f1(x) - bsval(x, korder, xknot, ncoef, bsclsq))
 end do                                      ! Вывод результата
! Выводим график сплайна, вычисленного CONFT
 call plotSpline(100, grdsiz, korder, xknot, ncoef, bscnft, grdsiz / 2.0)
 write(*, "(' Average error with BSLSQ fit: ', f8.5)") errlsq / 100.0
 write(*, "(' Average error with CONFT fit: ', f8.5)") errnft / 100.0

contains

! Вывод графика сплайна
! Используем режим векторного графа OM
 subroutine plotSpline(ndata, grdsiz, korder, xknot, ncoef, bscoef, dif)
! Текст plotSpline см. выше
 . . .
end program conftTest2                       ! Графики см. на рис. 1.12
```

Результат:

```
Average error with BSLSQ fit: 0.01814
Average error with CONFT fit: 0.01175
```



а *б*

Рис. 1.12. Сглаживание зашумленных данных, порожденных функцией $f(x) = 1/(1 + x^4)$: а - подпрограммой BSLSQ; б - подпрограммой CONFT

1.8.9. ПОДПРОГРАММА BSLS2 (DBSLS2)

Вычисляет по методу наименьших квадратов аппроксимирующий двумерный ТП-В-сплайн и возвращает его коэффициенты. Имеет вызов

CALL BSLS2(*nxdata, xdata, nydata, ydata, fdata, Ldf, kxord, kyord,* &
xknot, yknot, nxcoef, nycoef, xweigh, yweigh, bscoef)

Параметр *bscoef* является *выходным*. Остальные параметры - *входные*. Смысл параметров отражен в табл. 1.5 и 1.9.

Комментарий. При работе с BSLS2 могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	14	Низкая точность сглаживания; используйте, если возможно, двойную точность
4	5	Число повторяемости узлов не может превышать порядок сплайна
4	6	Узлы должны быть неубывающими
4	7	Все веса должны быть больше нуля
4	9	Абсциссы входных точек должны быть неубывающими
4	10	Наименьший элемент вектора *xdata* (*ydata*) должен быть не меньше узла *kordx* (*kordy*)
4	11	Наибольший элемент вектора *xdata* (*ydata*) должен быть не больше узла *nxcoef* + 1 (*nycoef* + 1)

Описание:

Подпрограмма BSLS2 вычисляет коэффициенты двумерного ТП-В-сплайна, решая нормальные уравнения в ТП-форме (см. [7, гл. 17] и [32]). Коэффициенты таковы, что минимизируется сумма

$$\sum_{i=1}^{n_y}\sum_{j=1}^{n_x} w_x(i)w_y(j)\left[\sum_{k=1}^{n_{cx}}\sum_{l=1}^{n_{cy}} b_{kl}B_{kl}(x_i,y_j) - f_{ij}\right]^2,$$

где функция B_{kl} - тензорное произведение двух В-сплайнов порядка k_x и k_y. В частности,

$$B_{kl}(x,y) = B_{kk_x t_x}(x)B_{lk_y t_y}(y).$$

Сплайн $\sum_{k=1}^{n_{cx}}\sum_{l=1}^{n_{cy}} b_{kl}B_{kl}$ можно оценить функцией BS2VL, а его частные производные - BS2DR.

Пример. Данные, подлежащие сглаживанию, порождаются функцией $e^x\sin(x + y) + \varepsilon$ на прямоугольной области $[0, 3]\times[0, 5]$. Здесь ε - случайная равномерно распределенная на отрезке $[-1, 1]$ величина. (График незашумленной исходной функции выведен на рис. 1.13.) Число точек равно 100×50; по оси x формируется кубический сплайн, по оси y - квадратичный. Полученный сплайн оценивается на сетке 3×5; оценки сравнивается с точными значениями.

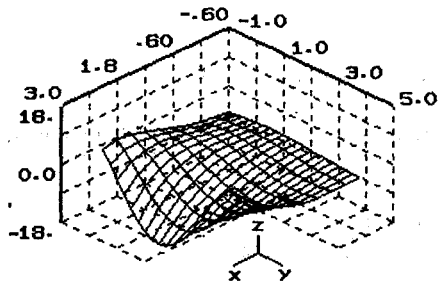


Рис. 1.13. Функция $e^x\sin(x + y)$ в области $[0, 3]\times[0, 5]$

```
program bsls2Test
 use dfimsl
 integer(4), parameter :: kxord = 4, kyord = 3, nxcoef = 15, nxdata = 100,          &
      nxvec = 4, nycoef = 7, nydata = 50, nyvec = 6, Ldf = nxdata
 integer(4) :: i, j, kb
 real(4) :: bscoef(nxcoef, nycoef), f, fdata(nxdata, nydata), rnoise,               &
      value(nxvec, nyvec), x, xdata(nxdata), xknot(nxcoef + kxord),                 &
      xvec(nxvec), xweigh(nxdata), y, ydata(nydata),                                &
      yknot(nycoef + kyord), yvec(nyvec), yweigh(nydata)
 f(x, y) = exp(x) * sin(x + y)              ! Задаем функцию
 call rnset(1234579)                        ! Затравка датчика случайных чисел
 do i = 1, nxcoef - kxord + 2               ! Задаем последовательности узлов по x и y
  xknot(i + kxord - 1) = 3.0 * (float(i - 1) / float(nxcoef - kxord + 1))
 end do
 xknot(nxcoef + 1) = xknot(nxcoef + 1) + 0.001
 do i = 1, nycoef - kyord + 2
  yknot(i + kyord - 1) = 5.0 * (float( i - 1) / float(nycoef - kyord + 1))
 end do
 yknot(nycoef+1) = yknot(nycoef + 1) + 0.001
 do i = 1, kxord - 1                        ! Совмещаем узлы
   xknot(i) = xknot(kxord)
   xknot(i + nxcoef + 1) = xknot(nxcoef + 1)
```

```
 end do
 do i = 1, kyord - 1
  yknot(i) = yknot(kyord)
  yknot(i + nycoef + 1) = yknot(nycoef + 1)
 end do
 ! Задаем сетку по x и y
 do i = 1, nxdata
  xdata(i) = 3.0 * (float(i - 1) / float(nxdata - 1))
 end do
 do i = 1, nydata
  ydata(i) = 5.0 * (float(i - 1) / float(nydata - 1))
 end do
 ! Задаем fdata в области [1, -1], добавляя к оценке функции случайный шум
 do i = 1, nydata
  do j = 1, nxdata
   rnoise = 2.0 * rnunf( ) - 1.0
   fdata(j, i) = f(xdata(j), ydata(i)) + rnoise
  end do
 end do
 xweigh = 1.0e0; yweigh = 1.0e0          ! Все веса равны 1.0
 ! Вычисляем коэффициенты ТП-В-сплайн методом наименьших квадратов
 call bsls2(nxdata, xdata, nydata, ydata, fdata, Ldf, kxord, kyord, xknot, yknot,          &
       nxcoef, nycoef, xweigh, yweigh, bscoef)
 ! Вывод заголовка
 write(*, "(13x, 'x', 14x, 'y', 10x, 'f(x, y)', 8x, 's(x, y)', 9x, 'Error')")
 ! Вывод результата на сетке [0, 3]×[0, 5].
 do i = 1, nxvec
  xvec(i) = float(i - 1)
 end do
 do i = 1, nyvec
  yvec(i) = float(i - 1)
 end do
 ! Оцениваем сплайн
 call bs2gd(0, 0, nxvec, xvec, nyvec, yvec, kxord, kyord, xknot, yknot,                    &
       nxcoef, nycoef, bscoef, value, nxvec)
 do i = 1, nxvec
  do j = 1, nyvec
   write(*, '(5f15.4)') xvec(i), yvec(j), f(xvec(i), yvec(j)),                             &
       value(i, j), (f(xvec(i), yvec(j)) - value(i, j))
  end do
 end do
end program bsls2Test
```

Результат:

x	y	f(x, y)	s(x, y)	Error
0.0000	0.0000	0.0000	0.2782	-0.2782
0.0000	1.0000	0.8415	0.7762	0.0653
0.0000	2.0000	0.9093	0.8203	0.0890
0.0000	3.0000	0.1411	0.1391	0.0020
0.0000	4.0000	-0.7568	-0.5705	-0.1863
0.0000	5.0000	-0.9589	-1.0290	0.0701
1.0000	0.0000	2.2874	2.2678	0.0196
1.0000	1.0000	2.4717	2.4490	0.0227
1.0000	2.0000	0.3836	0.4947	-0.1111
1.0000	3.0000	-2.0572	-2.0378	-0.0195
1.0000	4.0000	-2.6066	-2.6218	0.0151
1.0000	5.0000	-0.7595	-0.7274	-0.0321
2.0000	0.0000	6.7188	6.6923	0.0265
2.0000	1.0000	1.0427	0.8492	0.1935
2.0000	2.0000	-5.5921	-5.5885	-0.0035
2.0000	3.0000	-7.0855	-7.0955	0.0099
2.0000	4.0000	-2.0646	-2.1588	0.0942
2.0000	5.0000	4.8545	4.7339	0.1206
3.0000	0.0000	2.8345	2.5971	0.2373
3.0000	1.0000	-15.2008	-15.1079	-0.0929
3.0000	2.0000	-19.2605	-19.1698	-0.0907
3.0000	3.0000	-5.6122	-5.5820	-0.0302
3.0000	4.0000	13.1959	12.6659	0.5300
3.0000	5.0000	19.8718	20.5170	-0.6452

1.8.10. ПОДПРОГРАММА BSLS3 (DBSLS3)

Вычисляет по методу наименьших квадратов аппроксимирующий трехмерный ТП-В-сплайн и возвращает его коэффициенты. Имеет вызов

CALL BSLS3(*nxdata, xdata, nydata, ydata, nzdata, zdata, fdata,* &
Ldfdat, mdfdat, kxord, kyord, kzord, xknot, yknot, zknot, &
nxcoef, nycoef, nzcoef, xweigh, yweigh, zweigh, bscoef)

Параметр *bscoef* является *выходным*. Остальные параметры - *входные*. Смысл параметров отражен в табл. 1.5 и 1.9.

Комментарий. При работе с BSLS3 могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	13	Низкая точность сглаживания; используйте, если возможно, двойную точность
4	7	Число повторяемости узлов не может превышать порядок сплайна
4	8	Узлы должны быть неубывающими
4	9	Все веса должны быть больше нуля
4	10	Абсциссы входных точек должны быть неубывающими
4	11	Наименьший элемент вектора *xdata* (*ydata*) должен быть не меньше узла *kordx* (*kordy*)
4	12	Наибольший элемент вектора *xdata* (*ydata*) должен быть не больше узла *nxcoef* + 1 (*nycoef* + 1)

Описание:

Подпрограмма BSLS2 вычисляет коэффициенты трехмерного ТП-В-сплайна, решая нормальные уравнения в ТП-форме (см. [7, гл. 17] и [32]). Коэффициенты таковы, что минимизируется сумма

$$\sum_{i=1}^{n_x}\sum_{j=1}^{n_y}\sum_{p=1}^{n_z} w_x(i)w_y(j)w_y(p)\left[\sum_{k=1}^{n_{cx}}\sum_{l=1}^{n_{cy}}\sum_{m=1}^{n_{cz}} b_{klm}B_{klm}(x_i,y_j,z_p)-f_{ijp}\right]^2,$$

где функция B_{klm} - тензорное произведение трех В-сплайнов порядка k_x, k_y и k_z. В частности,

$$B_{klm}(x,y)=B_{kk_x t_x}(x)B_{lk_y t_y}(y)B_{uk_z t_z}(z).$$

Сплайн $\sum_{k=1}^{n_{cx}}\sum_{l=1}^{n_{cy}}\sum_{m=1}^{n_{cz}} b_{klm}B_{klm}(x,y,z)$ можно оценить функцией BS3VL, а его частные производные - BS3DR. Для получения значений на сетке можно употребить подпрограмму BS3GD.

1.9. СГЛАЖИВАЮЩИЕ КУБИЧЕСКИЕ СПЛАЙНЫ

1.9.1. ПЕРЕЧЕНЬ ПОДПРОГРАММ

Подпрограммы, вычисляющие сглаживающие кусочно-кубические сплайны, приведены в табл. 1.10.

Таблица 1.10. Подпрограммы, вычисляющие сглаживающие кубические сплайны

Подпрограмма	*Описание*
CSSED	Сглаживает одномерные данные путем обнаружения ошибки
CSSMH	Сглаживает зашумленные данные
CSSCV	Сглаживает зашумленные данные, используя перекрестную проверку

1.9.2. ПОДПРОГРАММА CSSED (DCSSED)

Сглаживает одномерные данные путем обнаружения ошибки. Имеет вызов

CALL CSSED(*ndata, xdata, fdata, dis, sc, maxit, sdata*)

Параметры подпрограммы CSSED:

Параметр *sdata* является *выходным*. Остальные параметры - *входные*. Смысл параметров, кроме приведенных ниже, отражен в табл. 1.4.

dis - коэффициент пропорциональности, используемый при определении расстояния сдвига по оси ординат точки, существенно отстоящей от интерполяционного сплайна, $0 < dis \le 1$. Рекомендуемое значение $dis = 1$.

sc - критерий остановки; $sc \ge 0$. Рекомендуемое значение $sc = 0$.

maxit - максимально допустимое число итераций.

sdata - вектор размера *ndata*, содержащий сглаженные данные.

Комментарии:

1. При работе может возникнуть информационная ошибка типа 1 с кодом 1, означающая, что достигнуто максимально допустимое число итераций.
2. Векторы *fdata* и *sdata* могут быть одним и тем же вектором.

Описание:

Подпрограмма CSSED предназначена для сглаживания слегка загрязненных данных. Как правило, подпрограмма неприменима, если более 25% данных содержат ошибки. Подпрограмма CSSED основана на алгоритме, приведенном в [33]. Порядок работы CSSED таков:

1) первоначально элементы вектора *xdata* упорядочивается по возрастанию их значений и принимается $s = f$;
2) затем подпрограммой CSAKM вычисляется интерполяционный кубический сплайн S_i для каждых шести точек входного набора данных (x_j, s_i), $j = i - 3, \ldots, i + 3$, $j \ne i$;
3) для каждого i-го интерполяционного сплайна вычисляется точечная энергия $pe_i = S_i(x_i) - s_i$;
4) вычисления завершаются, если число итераций превысило *maxit* или если
$$|pe_i| \le sc(x_{i+3} - x_{i-3}) / 6,\ i = 4, \ldots, n - 3.$$

Если приведенное неравенство нарушается, то изменяется i-й элемент вектора s: $s_i = s_i + d(pe_i)$, где $d = dis$. Заметьте, что первые и последние 3 точки не меняются. Таким образом, если эти точки содержат существенные ошибки, к полученному результату нужно подходить с известной осторожностью.

Если пользователь осведомлен о числе искаженных данных, то следует принять $d = 1$, $sc = 0$ и *maxit* установить равным числу искаженных данных. Если же такой информации нет, то следует задать $d = 0.5$, $maxit \le 2n$ и $sc = 0.5\ (\max(s) - \min(s))/(x_n - x_1)$. В любом случае полезно поэкспериментировать с названными величинами.

Пример. Функцией $f(x) = 5 + (5 + x^2 \sin x)/x$ на отрезке [1, 10] порождается 91 точка. Затем 10 значений искажаются данными вектора *rnoise*. Вслед при помощи CSSED выполняется сглаживание загрязненных данных. Первоначально значения параметров *dis* и *sc* задаются из предположения, что число искаженных данных неизвестно. Перед вторым вызовом CSSED считается, что есть информация о количестве искажений.

```
program cssedTest
 use dfimsl
 integer(4), parameter :: ndata = 91
 integer(4) :: i, maxit, isub(10) = (/ 6, 17, 26, 34, 42, 49, 56, 62, 75, 83 /)
 real(4) :: dis, f, fdata(91), sc, sdata(91), sin, x, xdata(91), rnoise(10)
 data rnoise / 2.5, -3.0, -2.0, 2.5, 3.0, -2.0, -2.5, 2.0, -2.0, 3.0 /
 ! Задаем функцию
 f(x) = (x * x * sin(x) + 5.0) / x + 5.0
 ! Полагаем, что число искаженных данных неизвестно
 dis = 0.5; sc = 0.56; maxit = 182
 xdata(1) = 1.0
 fdata(1) = f(xdata(1))
 do i = 2, ndata                          ! Задаем xdata и fdata
  xdata(i) = xdata(i - 1) + 0.1
  fdata(i) = f(xdata(i))
 end do
 ! Загрязняем данные
 do i = 1, 10
   fdata(isub(i)) = fdata(isub(i)) + rnoise(i)
 end do
 ! Сглаживаем данные
 call cssed(ndata, xdata, fdata, dis, sc, maxit, sdata)
 ! Вывод заголовка и данных
 write(*, "(' Case A - no specific information available' / ' f(x) f(x)+noise sdata(x)', /)")
 do i = 1, 10
   write(*, 3) f(xdata(isub(i))), fdata(isub(i)), sdata(isub(i))
 end do
 ! Полагаем, что число искаженных данных известно
 dis = 1.0; sc = 0.0; maxit = 10
 ! Сглаживаем данные. Будет выдано предупреждение из-за превышения maxit
```

```
call cssed(ndata, xdata, fdata, dis, sc, maxit, sdata)
! Вывод заголовка и данных
write(*,"(' Case B - specific information available', /, ' f(x) f(x) + noise sdata(x)', /)")
do i = 1, 10
  write(*, 3) f(xdata(isub(i))), fdata(isub(i)), sdata(isub(i))
end do
3 format(' ', f7.3, 8x, f7.3, 11x, f7.3)
end program cssedTest
```

Результат:

Case A - No specific information available

f(x)	f(x) + noise	sdata(x)
9.830	12.330	9.870
8.263	5.263	8.215
5.201	3.201	5.168
2.223	4.723	2.264
1.259	4.259	1.308
3.167	1.167	3.138
7.167	4.667	7.131
10.880	12.880	10.909
12.774	10.774	12.708
7.594	10.594	7.639

*** WARNING ERROR 1 from CSSED. Maximum number of iterations limit MAXIT
*** =10 exceeded. The best answer found is returned.

Case B - specific information available

f(x)	f(x) + noise	sdata(x)
9.830	12.330	9.831
8.263	5.263	8.262
5.201	3.201	5.199
2.223	4.723	2.225
1.259	4.259	1.261
3.167	1.167	3.170
7.167	4.667	7.170
10.880	12.880	10.878
12.774	10.774	12.770
7.594	10.594	7.592

1.9.3. ПОДПРОГРАММА CSSMH (DCSSMH)

Вычисляет одномерный кубический сплайн, сглаживающий зашумленные данные. Имеет вызов

CALL CSSMH(*ndata, xdata, fdata, weight, smpar, break, cscoef*)

Параметры подпрограммы CSSMH:

Параметры *break* и *cscoef* являются выходными. Остальные параметры - *входные*. Смысл параметров, кроме приведенных ниже, см. в табл. 1.4.

weight - вектор размера *ndata*, содержащий оценки стандартных отклонений (ошибок) данных *fdata*. Все элементы вектора *weight* должны быть положительны.

smpar - неотрицательное число, управляющее сглаживанием. Возвращаемый сплайн *s* таков, что

$$\sum_{i=1}^{n}\left|\frac{s(x_i)-f_i}{w_i}\right|^2 \le \sigma\,, \qquad (1.4)$$

где w = *weight*, σ = *smpar*. Рекомендуется, чтобы значение σ находилось в доверительном интервале левой части, т. е.

$$n-\sqrt{2n} \le \sigma \le n+\sqrt{2n}\,.$$

break - вектор размера *ndata*, содержащий точки разрыва КМ-представления кубического сплайна.

cscoef - матрица формы (4, *ndata*), содержащая коэффициенты отдельных кусков кубического сплайна.

Комментарии:

1. При работе с CSSMH могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	1	Достигнуто максимально допустимое число итераций. Возвращается наилучшая аппроксимация
4	3	Все элементы вектора *weight* должны быть больше нуля

2. Кубический сплайн можно оценить функцией CSVAL, а его производные - функцией CSDER.

Описание:

Подпрограмма CSSMH возвращает C^2-непрерывный естественный сглаживающий кубический сплайн, обеспечивающий минимум интеграла

$$\int_a^b s''(x^2)dx$$

при ограничениях, задаваемых (1.4). Рекомендуемое значение σ зависит от весов *w*.

Подпрограмма CSSMH основана на алгоритме, приведенном в [47], который также обсуждается в [7].

Пример. Данные, порождаемые функцией $f(x) = 1/(0.1 + 9(x - 1))^4)$, загрязняются случайным образом. Далее подпрограмма CSSMH пытается воспроизвести истинную функцию.

```
program cssmhTest
 use dfimsl
 integer(4), parameter :: ndata = 300
 integer(4) :: i
real(4) :: break(ndata), cscoef(4,ndata), error, f, fdata(ndata), fval, smpar = ndata,      &
      sval, weight(ndata), x, xdata(ndata), xt
 f(x) = 1.0 / (0.1 + (3.0 * (x - 1.0))**4) ! Задаем функцию и сетку
 do i = 1, ndata
  xdata(i) = 3.0 * (float(i - 1) / float(ndata - 1))
  fdata(i) = f(xdata(i))
 end do
 call rnset(1234579)                    ! Затравка датчика случайных чисел
 ! Загрязняем данные
 do i = 1, ndata
  fdata(i) = fdata(i) + 2.0 * rnunf( ) - 1.0
 end do
 weight = 1.0 / sqrt(3.0)               ! Задаем вектор весов и сглаживаем данные
 call cssmh(ndata, xdata, fdata, weight, smpar, break, cscoef)
 ! Вывод заголовка и 10 значений результата
 write(*, "(12x, 'x', 9x, 'Function', 7x, 'Smoothed', 10x, 'Error')")
 do i = 1, 10
  xt = 90.0 * (float(i - 1) / float(ndata - 1))
  ! Оцениваем сплайн
  sval = csval(xt, ndata - 1, break, cscoef)
  fval = f(xt)
  error = sval - fval
  write(*, '(4f15.4)') xt, fval, sval, error
 end do
end program cssmhTest
```

Результат:

x	Function	Smoothed	Error
0.0000	0.0123	0.1118	0.0995
0.3010	0.0514	0.0646	0.0131
0.6020	0.4690	0.2972	-0.1718
0.9030	9.3312	8.7022	-0.6289
1.2040	4.1611	4.7887	0.6276
1.5050	0.1863	0.2718	0.0856
1.8060	0.0292	0.1408	0.1116
2.1070	0.0082	0.0826	0.0743
2.4080	0.0031	0.0076	0.0045
2.7090	0.0014	-0.1789	-0.1803

1.9.4. ПОДПРОГРАММА CSSCV (DCSSCV)

Вычисляет одномерный кубический сплайн, сглаживающий зашумленные данные, используя для оценки сглаживающего параметра перекрестную проверку. Имеет вызов

CALL CSSCV(*ndata, xdata, fdata, iequal, break, cscoef*)

Параметры подпрограммы CSSCV:

Параметры *break* и *cscoef* являются выходными. Остальные параметры - *входные*. Смысл параметров, кроме *iequal*, см. в табл. 1.4 и в предшествующем разделе. Заметим, что *ndata* > 2.

iequal - флаг, говорящий подпрограмме о том, что данные размещены равномерно.

Комментарий. При работе может возникнуть информационная ошибка типа 4 с кодом 2, означающая, что в *xdata* есть совпадающие значения.

Описание:

Подпрограмма CSSCV возвращает C^2-непрерывный естественный сглаживающий кубический сплайн s_σ, обеспечивающий минимум интеграла

$$\int_a^b s_\sigma''(x^2)dx$$

при ограничениях, задаваемых неравенством

$$\sum_{i=1}^{n}\left|s_\sigma(x_i)-f_i\right|^2 \le \sigma,$$

где σ - сглаживающий параметр. В отличие от приведенной выше подпрограммы CSSMH, которой нужно передать значение сглаживающего параметра σ, настоящая подпрограмма пытается найти оптимальное значение σ, используя технику перекрестных проверок. То есть σ выбирается таким, что сглаживающий сплайн s_σ наилучшим образом аппроксимирует данные в точке x_i, если он вычислен с использованием всех данных, кроме i-го значения; это справедливо для всех $i = 1, \ldots, n$. Алгоритм рассмотрен в [21].

Пример. Решается та же, что и в примере для CSSMH, задача. Поскольку данные расположены равномерно, задаем *iequal* = 1. Код практически тот же, что и в примере для CSSMH. Вместо вызова CSSMH пишем

```
call csscv(ndata, xdata, fdata, iequal, break, cscoef)
```

Результат (приводится для сравнения с решением, полученным CSSMH):

x	F*unction*	Smoothed	Error
0.0000	0.0123	0.2528	0.2405
0.3010	0.0514	0.1054	0.0540
0.6020	0.4690	0.3117	-0.1572
0.9030	9.3312	8.9461	-0.3850
1.2040	4.1611	4.6847	0.5235
1.5050	0.1863	0.3819	0.1956
1.8060	0.0292	0.1168	0.0877
2.1070	0.0082	0.0658	0.0575
2.4080	0.0031	0.0395	0.0364
2.7090	0.0014	-0.2155	-0.2169

1.10. ПРИБЛИЖЕНИЕ ЧЕБЫШЕВА. ПОДПРОГРАММА RATCH (DRATCH)

Вычисляет рациональное взвешенное приближение Чебышева

$$R_m^n = \frac{\sum_{i=1}^{n+1} P_i \phi^{i-1}(x)}{\sum_{i=1}^{m+1} Q_i \phi^{i-1}(x)}$$

непрерывной функции $f(x)$ на заданном отрезке. Имеет вызов

CALL RATCH(*f, phi, weight, a, b, n, m, p, q, error*)

Параметры подпрограммы RATCH:

Пользовательские функции: f, phi, weight.

Входные: a, b, n, m.

Выходные: p, q, error.

f - пользовательская, подлежащая аппроксимации функция. Имеет вид *f*(*x*), где *x* - независимая переменная.

phi - пользовательская функция формы *phi*(*x*), задающая функцию ϕ(*x*) в дробно-рациональном выражении R_m^n.

weight - пользовательская функция вида *weight*(*x*), масштабирующая максимальную ошибку. Должна быть непрерывной и отличной от нуля на отрезке [*a, b*].

Замечание. Пользовательские функции *f, phi* и *weight* должны быть снабжены в вызывающей программе атрибутом EXTERNAL.

a, b - соответственно левая и правая границы аппроксимации.

n, m - соответственно степени числителя и знаменателя приближения.

p, q - векторы размера *n* + 1, содержащие соответственно коэффициенты числителя и знаменателя.

error - минимаксная ошибка аппроксимации.

Комментарий. При работе с RATCH могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	1	Достигнуто максимально допустимое число итераций *itmax*. Можно вместо RATCH вызвать подпрограмму R2TCH и установить большее значение *itmax*
3	2	Ошибка уменьшена до предела. Векторы *p* и *q* содержат хорошую аппроксимацию, однако все же не обеспечивающую приближение Чебышева
4	3	Линейная система, задаваемая векторами *p* и *q*, алгоритмически вырожденная. Возможно, аппроксимация непригодна
4	4	Последовательность критических точек немонотонна. Возможно, аппроксимация непригодна
4	5	Величина ошибки в некоторых критических точках чрезмерно велика. Возможно, рациональная функция имеет полюса
4	6	Весовая функция *weight* не может быть равной нулю на отрезке [*a, b*]

Описание:

Подпрограмма RATCH создана для вычисления наилучшей взвешенной L_∞-аппроксимации Чебышева заданной функции, возвращаемой в виде дробно-рационального выражения R_m^n. Цель выполняемых подпрограммой RATCH вычислений - найти коэффициенты, обеспечивающие минимум взвешенной ошибки

$$\left\| \frac{f - R_m^n}{w} \right\| = \max_{x \in [a,b]} \frac{\left| f(x) - \frac{\sum_{i=1}^{n+1} P_i \phi^{i-1}(x)}{\sum_{i=1}^{m+1} Q_i \phi^{i-1}(x)} \right|}{w(x)}.$$

Заметьте, что задание $\phi(x) = x$ приводит к обычному рациональному приближению.

Алгоритм, реализованный в подпрограмме, реализует наилучшее равномерное приближение на отрезке $[a, b]$ к непрерывной функции $f(x)$. Это означает, что приближение имеет ровно $n + m + 2$ равновесных колебания. То есть существуют $n + m + 2$ точки $t_1 < \ldots < t_{n+m+2}$, удовлетворяющие условию

$$e(t_i) = -e(t_{i+1}) = \pm \left\| \frac{f - R_m^n}{w} \right\|.$$

Такие точки называются *альтернансом*. Есть, однако, много примеров, когда лучшее рациональное приближение заданной функции имеет меньше или больше, чем $n + m + 2$, точек альтернанса. В этом случае возвращенный подпрограммой RATCH результат может оказаться неудовлетворительным.

Подпрограмма RATCH основана на работе [19]. Достаточно полно проблема построения рационального приближения Чебышева рассмотрена в работе [18].

Пример. Вычисляется наилучшая рациональная аппроксимация гамма-функции G на отрезке [2, 3], с весовой функцией $w = 1$ и $n = m = 2$. Выводятся максимальная ошибка и коэффициенты. Применяется двойная точность. Задача взята из [19].

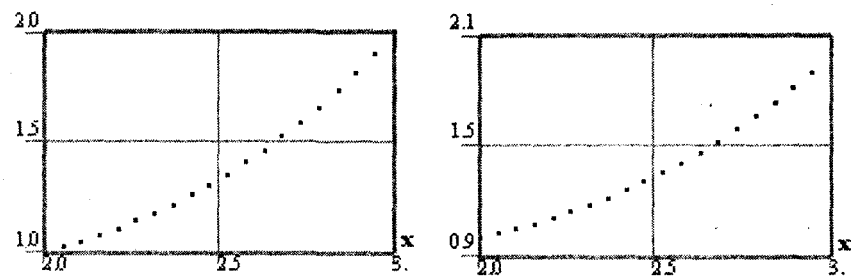
```
program ratchTest
 use dfimsl
 integer(4), parameter :: m = 2, n = 2, nxy = 20
 integer(4) :: i
 real(8) :: a, b, error, p(n + 1), q(m + 1)
 real(8), external :: f, phi, weight
 ! Массив xyvalues введен для вывода графика аппроксимации
```

```
real(kind(1e0)), allocatable :: xyvalues(:, :)
!dec$attributes array_visualizer :: xyvalues
a = 2.0d0; b = 3.0d0                    ! Вычисляем рациональное приближение
call dratch(f, phi, weight, a, b, n, m, p, q, error)
! Вывод p, q и минимаксной ошибки
write(*, '(1x, a)') 'In double precision we have:'
write(*, 1) 'p = ', p
write(*, 1) 'q = ', q
write(*, 1) 'Error = ', error
1 format(' ', a, 5x, 3f20.12, /)
allocate(xyvalues(2, nxy))
do i = 1, nxy                           ! Вывод гамма-функции
 xyvalues(1, i) = a + float(i - 1) / float(nxy - 1)
 xyvalues(2, i) = f(dble(xyvalues(1, i)))
end do
call vGraph(xyvalues, nxy)              ! Результат см. на рис. 1.14, а
deallocate(xyvalues)
allocate(xyvalues(2, nxy))
! Вывод рационального приближения Чебышева
do i = 1, nxy
 xyvalues(1, i) = a + float(i - 1) / float(nxy - 1)
 xyvalues(2, i) = fx(dble(xyvalues(1, i)), p, n) / fx(dble(xyvalues(1, i)), q, m)
end do
call vGraph(xyvalues, nxy)              ! Результат см. на рис. 1.14, б
deallocate(xyvalues)
end program ratchTest

function f(x)
 real(8) :: f, x
 real(8), external :: dgamma
 f = dgamma(x)
end function f

function phi(x)
 real(8) :: phi, x
 phi = x
end function phi

function weight(x)
 real(8) :: weight, x
 real(8), external :: dgamma
 weight = dgamma(x)
end function weight
```

```
function fx(x, coeff, ndeg)                ! Оценивает многочлен по схеме Горнера
 integer(4) :: ndeg, i                     ! Описание схемы см., например, в [5, с. 12]
 real(8) :: fx, x, coeff(ndeg + 1)
 fx = coeff(ndeg + 1)
 do i = ndeg, 1, -1
  fx = coeff(i) + x * fx
 end do
end function fx
```

Результат:

```
In double precision we have:
p = 1.265583562487 -0.650585004466 0.197868699191
q = 1.000000000000 -0.064342721236 -0.028851461855
Error = -0.000026934190
```



а *б*

Рис. 1.14. Гамма-функция на отрезке [2, 3] и ее приближение:
а - гамма-функция; б - рациональное приближение гамма-функции

2. АППРОКСИМАЦИЯ КРИВЫХ И ПОВЕРХНОСТЕЙ СПЛАЙНАМИ БИБЛИОТЕКИ IMSL 90 MP

2.1. ОБЩИЕ СВЕДЕНИЯ

Приводятся описания процедур, выполняющих аппроксимацию (приближение или сглаживание) дискретного набора данных в виде B-сплайнов (в одномерном случае) или их тензорного произведения (двумерный случай). В основу процедур положены данные в [7] алгоритмы. Задача аппроксимации решается как взвешенная задача наименьших квадратов. В двумерном случае осуществляется регуляризация задачи наименьших квадратов, причем заданные по умолчанию значения параметров могут быть пользователем изменены. Также имеется возможность изменять форму результирующих кривых и поверхностей на основе информации, которая не может быть легко отражена в задаче наименьших квадратов. Так, пользователь может получить монотонно убывающую кривую, расположенную выше оси $y = 0$ и имеющую определенные знаки второй производной на заданных подынтервалах. Решение подобной задачи иллюстрируется примером 3. Пример 4 демонстрирует способ построения 3D-сплайнов в виде поверхностей с неотрицательной z-координатой. Для вывода сплайнов используется режим векторного графа отображателя массивов (ОМ).

2.1.1. СПЛАЙНЫ НА ПЛОСКОСТИ

2.1.1.1. Порядок построения плоских сплайнов

Для приближения или сглаживания следует:

1) задать степень сплайна и его узлы. Используйте описанный в модуле MP_TYPES производный тип *s_spline_knots* или *d_spline_knots* для объявления переменных, передаваемых в качестве параметров сглаживающей процедуры. Названные производные типы данных обсуждаются ниже;

2) выбрать ограничения, которым сплайны должны удовлетворять. Используйте родовую, производного типа функцию SPLINE_CONSTRAINTS для задания ограничений (в общем случае необязательных), которые будут переданы сглаживающей процедуре. Производный тип, который имеет функция SPLINE_CONSTRAINTS, обсуждается ниже;

3) определить данные, на основе которых строятся сплайны. Данные задаются в виде пары независимых или зависимых величин

(x_i, y_i), $i = 1, ..., ndata$

и дисперсии: каждая зависимая переменная требует оценки ее дисперсии σ_i;

4) использовать массивоподобную функцию SPLINE_FITTING для вычисления коэффициентов B-сплайна, передаваемых оценщику SPLINE_VALUES;
5) выполнить, употребляя функцию SPLINE_VALUES, принимающую возвращенные функцией SPLINE_FITTING коэффициенты, оценку сплайна, его производной и квадратный корень его дисперсии.

2.1.1.2. Производные типы данных s_knots и d_knots

Степень сплайна и его узлы задаются переменной, имеющей тип

```
type ?_spline_knots
  integer spline_degree
  real(kind(?)), pointer :: ?_knots(:)
end type
```

в котором ?_ - это *s_* или *d_*, а ? - это 1.0e0 или 1.0d0, употребляемые соответственно для одинарной или двойной точности. Определение типа дано в модуле MP_TYPES, ссылка на который имеется в модуле SPLINE_FITTING_INT. Примеры объявлений и определений переменных рассмотренного типа даны ниже.

Замечание. Приведенная интерпретация пары символов ?_ и отдельного символа ? распространяется на все даваемые ниже описания производных типов данных.

2.1.1.3. Функция SPLINE_CONSTRAINTS

Задание ограничений, действующих в определенной точке, осуществляется массивом производного типа. Каждый элемент этого массива имеет компоненты, описываемые типом

```
type ?_spline_constraints
  integer derivative_index
  real(kind(?)) where_applied
  character(len = *) constraint_indicator
  real(kind(?)) value_applied
end type
```

Обладающая родовым интерфейсом функция SPLINE_CONSTRAINTS описана в модуле SPLINE_FITTING_INT. Функция возвращает в зависимости от используемой точности массив типа *s_spline_constraints* или *d_spline_constraints*.

2.1.1.4. Оценочная функция SPLINE_VALUES

После вычисления коэффициентов B-сплайна оценка сплайна (значения его *y*-координат в заданных на оси абсцисс точках), его производной или квадратного корня дисперсии возвращаются функцией SPLINE_VALUES. Одно из назначений функции - подготовить данные для графического отображения. Функция принимает вектор значений *x*-координат и возвращает вектор того же размера, что и входной, с рассчитанными данными. Также перечисленные значения могут быть вычислены и для одной точки.

2.1.1.5. Массивоподобная функция SPLINE_FITTING

Коэффициенты B-сплайна возвращаются родовой функцией SPLINE_-FITTING. Тип коэффициентов определяется используемой для входных данных точностью (одинарной или двойной). Функция принимает массив с данными и переменную типа *?_spline_knots*. Массив производного типа *?_spline_constraints* является необязательным параметром функции.

2.1.2. ПРОСТРАНСТВЕННЫЕ СПЛАЙНЫ

2.1.2.1. Порядок выполнения двумерного сглаживания

Для двумерного сглаживания следует:

1) задать степень сплайна и его узлы по двум измерениям. Степень сплайна должна быть одной и той же по каждому измерению. Используйте описанный в модуле MP_TYPES производный тип *s_spline_knots* или *d_spline_knots*. Этот же тип употребляется и для плоских сплайнов, но для пространственного сглаживания необходимо задать значения по двум измерениям;
2) выберать параметры регуляризации и ограничения, которым должен удовлетворять сплайн (сплайн-функция, получаемая как тензорное произведение сплайнов). Значения параметров регуляризации передаются сглаживающей процедуре посредством производного типа *s_options* или *d_options*. На практике для получения удовлетворительных результатов важно правильно задать параметры разреженности и, возможно, пологости (малой кривизны) или удаленности, появляющихся в модели наименьших квадратов;
3) использовать родовую, производного типа функцию SURFACE_-CONSTRAINTS для задания ограничений (в общем случае необязательных), которые будут переданы сглаживающей процедуре. Производный тип, который имеет функция SURFACE_CONSTRAINTS, обсуждается ниже;
4) определить данные, на основе которых строятся сплайны. Данные задаются в виде тройки независимых или зависимых от одной переменной величин

(x_i, y_i, z_i), $i = 1, ..., ndata$

и дисперсии: каждая зависимая переменная требует оценки ее дисперсии σ_i;

5) использовать массивоподобную функцию SURFACE_FITTING для вычисления коэффициентов B-сплайна (тензорного произведения), передаваемых оценщику SURFACE_VALUES;

6) выполнить, употребляя функцию SURFACE_VALUES, принимающую возвращенные функцией SURFACE_FITTING коэффициенты, оценку сплайна, его производной и квадратный корень его дисперсии.

2.1.2.2. Функция SURFACE_CONSTRAINTS

Задание действующих в определенной точке ограничений поверхностного B-сплайна, являющегося тензорным произведением плоских B-сплайнов (ТП-B-сплайнов), осуществляется массивом производного типа. Каждый элемент этого массива имеет компоненты, описываемые типом

```
type ?_surface_constraints
  integer derivative_index(2)
  real(kind(?)) where_applied(2)
  character(len = *) constraint_indicator
  real(kind(?)) value_applied
  real(kind(?)) periodic_point(2)
end type
```

Обладающая родовым интерфейсом функция SURFACE_CONSTRAINTS описана в модуле SPLINE_FITTING_INT. Функция возвращает в зависимости от используемой точности массив типа *s_surface_constraints* или *d_surface_constraints*.

2.1.2.3. Оценочная функция SURFACE_VALUES

После вычисления коэффициентов ТП-B-сплайна оценка сплайна (значения его z-координат в заданных на плоскости $x0y$ точках), его производной или квадратного корня дисперсии возвращается функцией SURFACE_VALUES. Одно из назначений функции - подготовить данные для графического отображения. Функция принимает векторы значений x- и y-координат и возвращает вектор, размер которого равен произведению размеров входных векторов, с рассчитанными данными. Также перечисленные значения могут быть вычислены и для одной точки.

2.1.2.4. Массивоподобная функция SURFACE_FITTING

Коэффициенты ТП-В-сплайна возвращаются родовой функцией SURFACE_FITTING. Тип коэффициентов определяется используемой для входных данных точностью (одинарной или двойной). Функция принимает массив с данными и переменные типа ?*_spline_knots* для *x*- и *y*-координат. Массив производного типа ?*_surface_constraints* является необязательным параметром функции.

2.2. ОПИСАНИЕ ФУНКЦИЙ, УПОТРЕБЛЯЕМЫХ С ПЛОСКИМИ СПЛАЙНАМИ

2.2.1. ФУНКЦИЯ SPLINE_CONSTRAINTS

Функция возвращает массив производного типа ?*_spline_constraints*. Синтаксис вызова функции с ключевыми словами для узла с номером *j*:

```
?_spline_constraints(j) =                                          &
    SPLINE_CONSTRAINTS([derivative = derivative_index,]            &
    point = where_applied, [value = value_applied,],               &
    type = constraint_indicator, [periodic_point = periodic_point])
```

Указанные в квадратных скобках параметры являются необязательными. Причем для каждого ограничения должен быть задан параметр *value* = или *periodic_point* =. Но нельзя задавать эти параметры одновременно.

Все параметры функции, как обязательные, так и необязательные, являются *входными*. Тип числовых параметров - REAL(4) или REAL(8).

Обязательные параметры функции SPLINE_CONSTRAINTS:

point = where_applied - точка в интервале данных, в которой должны быть выполнены задаваемые ограничения.

type = constraint_indicator - вид ограничения для сплайна или его производной. Может быть задан символами '==', '<=', '>=', '.=.' или '.=-', означающими соответственно, что значение сплайна или его производной:

- равны;
- не больше чем;
- не меньше чем;
- равны значению сплайна в другой точке;
- равны взятому с обратным знаком значению сплайна в другой точке.

Последние два ограничения называются соответственно *периодическим* и *отрицательно-периодическим*. Пример 4 содержит ограничения такого рода.

Необязательные параметры функции SPLINE_CONSTRAINTS:

derivative = derivative_index - номер производной формируемого сплайна, которая должна удовлетворять задаваемым ограничениям. При этом значение 0 соответствует сплайн-функции, значение 1 - первой производной и т. д. Если параметр опущен, то *derivative* = 0, т. е. ограничению должна отвечать сплайн-функция.

periodic_point = periodic_point - автоматическое определение значения второй независимой переменной для периодических ограничений.

2.2.2. ФУНКЦИЯ SPLINE_VALUES

Функция возвращает в качестве результата вектор, отвечающий входному вектору *variables*. Использует в качестве необязательного параметра матрицу ковариаций, когда вычисляется квадратный корень дисперсии. Результат будет скаляром, если *variables* - скаляр.

Функция имеет следующий вызов

```
yvalues = SPLINE_VALUES(derivative = derivative,                    &
    variables = variables, knots = knots,                           &
    coeffs = c, [covariance = G], [iopt = iopt])
```

Все параметры функции являются *входными*. Тип числовых параметров - REAL(4) или REAL(8). Размер вектор-результата равен размеру вектора *variables*.

Обязательные параметры функции SPLINE_VALUES:

derivative = derivative - номер вычисляемой производной. Не может быть меньше нуля. Для оценки сплайн-функции используется нуль.

variables = variables - значения независимых переменных, при которых выполняется оценка сплайна или его производной. В качестве *variables* употребляется либо вектор (массив ранга 1), либо скаляр.

knots = knots - переменная производного типа ?*_spline_knots*, используемая для получения массива *coeffs* функцией SPLINE_FITTING. Содержит степень сплайна, число узлов и сами узлы.

coeffs = c - коэффициенты в представлении сплайн-функции

$$f(x) = \sum_{j=1}^{n} c_j B_j(x) . \qquad (2.1)$$

Коэффициенты возвращаются в результате вызова

c = spline_fitting(...)

порядок выполнения которого приводится ниже. Причем величина n = SIZE(c) удовлетворяет соотношению

$$n - 1 + spline\_degree = \text{SIZE}(?\_knots), \qquad (2.2)$$

в котором *spline_degree* и *?_knots* - компоненты параметра *knots*, имеющего тип *?_spline_knots*.

Необязательные параметры функции SPLINE_VALUES:

covariance = G - параметр, если присутствует, равен квадратному корню дисперсии функции:

$$e(x) = \sqrt{b(x)^T G b(x)},$$

где

$$b(x) = (B_1(x), ..., B_n(x))^T,$$

а G - матрица ковариации, связанная с коэффициентами сплайна

$$c = (c_1, ..., c_n)^T.$$

Параметр G является необязательным. Он возвращается функцией SPLINE_FITTING, обсуждаемой ниже. Когда вычисляется квадратный корень функции дисперсии, параметры *derivative* и *c* не используются.

iopt = *iopt* - необязательный параметр производного типа *?_options*. В настоящей версии не используется.

2.2.3. ФУНКЦИЯ SPLINE_FITTING

Возвращает взвешенное приближение функции B-сплайном (2.1), найденное по дискретным одномерным данным с использованием метода наименьших квадратов. Ограничения на сплайн и его производные могут быть опущены. После выполнения приближения можно, применив SPLINE_VALUES, оценить сплайн-функцию, ее производные или квадратный корень дисперсии. Функция SPLINE_FITTING имеет следующий вызов:

c = SPLINE_FITTING(*data* = *data*(1:3, :), *knots* = *knots*, &
constraints = *spline_constraints*, [*covariance* = *G*], [*iopt* = *iopt*])

Параметры функции SPLINE_FITTING:

Входные: knots, constraints.

Входные/выходные: data, iopt.

Выходной: covariance.

Функция возвращает массив коэффициентов сплайна, размер которого вычисляется по формуле (2.2).

Обязательные параметры:

data = *data*(1:3, :) - перенимающий форму массив, содержащий данные в следующем порядке: *data*(1, *i*) = x_i, *data*(2, *i*) = y_i и *data*(3, *i*) = σ_i, *i* = 1,..., *ndata*. Если дисперсии σ_i неизвестны, но пропорциональны неизвестной величине, то можно задать *data*(3, *i*) = 1, *i* = 1, ..., *ndata*. Параметр имеет тип REAL(4) или REAL(8).

knots = *knots* - параметр производного типа ?_*spline_knots*, определяющий степень сплайна и его узлы на интервале приближения.

Необязательные параметры:

constraints = *spline_constraints* - вектор производного типа ?_*spline_constraints*, задающий ограничения, которым сплайн должен удовлетворять. Тип параметра совпадает с типом массива *data*.

covariance = *G* - перенимающий форму массив ранга 2, имеющий ту же точность, что и массив *data*. Содержит на выходе матрицу ковариаций коэффициентов. Используется для вычисления квадратного корня дисперсии.

iopt = *iopt*(:) - вектор производного типа. Используется для пересылки необязательных данных функции SPLINE_FITTING. Работает с приведенными в табл. 2.1 опциями.

Таблица 2.1. Опции, употребляемые с параметром iopt

Опция	*Значение опции*
spline_fitting_tol_equal	1
spline_fitting_tol_least	2

iopt(*io*) = ?_options(*spline_fitting_tol_equal*, ?_*value*) - устанавливает значение для определения ситуации, когда ограничения равенства имеют дефицит ранга. Значение по умолчанию ?_*value* = 10^{-4}.

iopt(*io*) = ?_options(*spline_fitting_tol_least*, ?_*value*) - устанавливает значение для определения ситуации, когда уравнения в модели наименьших квадратов имеют дефицит ранга. Значение по умолчанию ?_*value* = 10^{-4}.

Описание:

Процедура аналогична процедуре CONFT (DCONFT) библиотеки IMSL 77. Дополнительно вычисляется квадратный корень дисперсии, но не поддерживаются ограничения для интеграла сплайн-функции. Матрица проблемы наименьших квадратов имеет ширину ленты, равную порядку сплайна. Это обстоятельство позволяет применить более эффективный, чем в CONFT (DCONFT), алгоритм в случае, когда отсутствуют ограничения.

Когда ограничения заданы, процедура решает проблему наименьших квадратов с ограничениями типа равенства и неравенства. Результатом решения проблемы является ленточная верхняя треугольная матрица, на основе которой находятся коэффициенты элементарных сплайнов, образующих в итоге составную сплайновую кривую. Используемый алгоритм позволяет решать неполноранговую задачу наименьших квадратов. Его описание можно найти в [34] и [38]. В процедуре CONFT (DCONFT) библиотеки IMSL 77 применяется алгоритм QPROG, в котором модель наименьших квадратов должна обладать полным рангом.

Замечание. Сообщения о завершающих ошибках находятся в файле messages.gls. Они имеют номера 1340-1367.

Пример 1. Выполняется интерполяция (приближение) кубическим сплайном функции

$$g(x) = e^{-x^2/2}$$

на сетке, заданной точками

$$x_i = (i - 1)\Delta x,\ i = 1, ..., ndata.$$

Сплайн удовлетворяет условиям

$$f(x_i) = g(x_i),\ i = 0,\ \ldots,\ ndata;$$
$$\frac{d^2 f}{dx^2}(x_i) = \frac{d^2 g}{dx^2}(x_i),\ i = 0,\ i = ndata. \qquad (2.3)$$

Программа также вычисляет оценку *diff* максимального отклонения результата от узлов сплайна. График полученного интерполяционного сплайна строится OM в виде векторного графа.

```
program ex1
 use spline_fitting_int
 use show_int
 use norm_int
 implicit none
 integer(4) :: i
 integer(4), parameter :: ndata = 24, nord = 4, ndegree = nord - 1,          &
      nbkpt = ndata + 2 * ndegree, ncoeff = nbkpt - nord, nvalues = 2 * ndata
 real(kind(1e0)), parameter :: zero = 0e0, one = 1e0, half = 5e-1
 real(kind(1e0)), parameter :: delta_x = 0.15, delta_xv = 0.4 * delta_x
 real(kind(1e0)), target :: xdata(ndata), ydata(ndata),                      &
      spline_data(3, ndata), bkpt(nbkpt), ycheck(nvalues), coeff(ncoeff), diff
 ! Массив xyvalues введен для вывода графика сплайна
 real(kind(1e0)), allocatable :: xyvalues(:, :)
 !dec$attributes array_visualizer :: xyvalues
```

```
 real(kind(1e0)), pointer :: pointer_bkpt(:)
 type(s_spline_knots) :: break_points
 type(s_spline_constraints) :: constraints(2)
 allocate(xyvalues(2, nvalues))
 xdata = (/ ((i - 1) * delta_x, i = 1, ndata) /)
 ydata = exp(-half * xdata**2)
 ! x-координаты узлов сплайна
 xyvalues(1, :) =(/ (0.03 + (i - 1) * delta_xv, i = 1, nvalues) /)
 ! Вектор, используемый для проверки результата
 ycheck= exp(-half * xyvalues(1, :)**2)
 spline_data(1, :) = xdata
 spline_data(2, :) = ydata
 spline_data(3, :) = one
 ! Задаем узлы сплайна
 bkpt(1:ndegree) = (/ (i * delta_x, i = -ndegree, -1) /)
 bkpt(nord:nbkpt - ndegree) = xdata
 bkpt(nbkpt - ndegree + 1:nbkpt) = (/ (xdata(ndata) + i * delta_x, i = 1, ndegree) /)
 ! Присоединяем ссылку к сформированным выше данным (вектору bkpt)
 ! и задаем степень и узлы сплайна
 pointer_bkpt => bkpt
 break_points = s_spline_knots(ndegree, pointer_bkpt)
 ! Задаем традиционные для кубических интерполяционных сплайнов ограничения
 constraints(1) = spline_constraints(derivative = 2, point = bkpt(nord), type = '==', value = -one)
 constraints(2) = spline_constraints(derivative = 2, point = bkpt(nbkpt - ndegree),      &
       type = '==', value = (-one + xdata(ndata)**2) * ydata(ndata))
 coeff = spline_fitting(data = spline_data, knots = break_points, constraints = constraints)
 ! y-координаты оценки сплайна в его узлах
 xyvalues(2, :) = spline_values(0, xyvalues(1, :), break_points, coeff)
 ! Функция NORM вернет ∞-норму вектора xyvalues(2, :) - ycheck
 diff = norm(xyvalues(2, :) - ycheck, huge(1)) / delta_x**nord
 if(diff <= one) print *, 'Example 1 for SPLINE_FITTING is correct.'
 ! Для вывода интерполяционного сплайна используем режим векторного графа OM
 call vGraph(xyvalues, nvalues)
 deallocate(xyvalues)
end program ex1

subroutine vGraph(fun, nvalues)          ! Выводит массив как векторный граф OM
 use avdef
 use avviewer
 use dflib
 integer(4) :: nvalues
 real(4) :: fun(2, nvalues)
 integer(4) hv, status, nError
```

```
character(1) :: key
character(av_max_label_len) :: xLabel = 'x'
call faglStartWatch(fun, status)          ! Сообщаем ОМ имя отображаемого массива
print *, "Starting Array Viewer"
! Запускаем ОМ с использованием fav-подпрограммы
call favStartViewer(hv, status)
if(status /= 0) then
 call favGetErrorNo(hv, nError, status)
 if(nError /= 0) then
  print *, "Array Viewer reports error ", nError
  stop
 end if
end if
! Передаем ОМ данные подлежащего отображению массива
call favSetArray(hv, fun, status)
! Задаем заголовок экземпляра ОМ
call favSetArrayName(hv, "Spline 1", status)
! Отображаем массив в виде векторного графа
call favSetGraphType(hv, VectorGraph, status)
! Задаем режим вывода заданных пользователем имен осей координат
call favSetUseAxisLabel(hv, x_axis, 1, status)
! Новое (вместо dim1) имя x-оси координат. Длина переменной,
! задающей имя оси, равна AV_MAX_LABEL_LEN
call favSetAxisLabel(hv, x_axis, xLabel, status)
! Показываем ОМ на экране
call favShowWindow(hv, av_true, status)
print *, "Press any key to close down the viewer"
key = getcharqq( )
call favEndViewer(hv, status)             ! Закрываем ОМ
call faglEndWatch(fun, status)            ! Освобождаем ресурсы
end subroutine vGraph                     ! Результат приведен на рис. 2.1
```

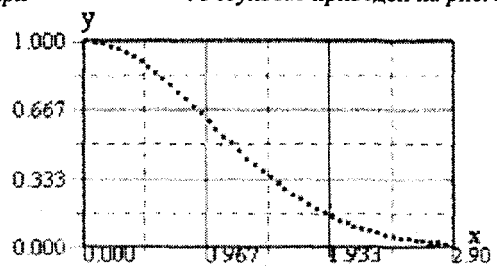


Рис. 2.1. Аппроксимация функции $g(x) = e^{-x^2/2}$ *интерполяционным кубическим сплайном*

Пример 2. Функция

$$g(x) = e^{-x^2/2}(1 + noise)$$

аппроксимируется сплайном на равномерной сетке

$$x_i = (i - 1)\Delta x,\ i = 1, ..., ndata.$$

Шум *noise* представляется равномерно распределенными случайными числами из интервала $[-\tau, \tau]$, где $\tau = 0.01$. Задаваемые ограничения таковы:

- кривая выпукла вверх на отрезке $0 \le x \le 1$ (имеет на нем неположительную вторую производную) и выпукла вниз на отрезке $1 < x \le 4$ (имеет на этом отрезке неотрицательную вторую производную);
- вторая производная точно равна нулю при $x = 1$ (т. е. $x = 1$ является точкой перегиба кривой);
- первая производная равна нулю при $x = 0$;
- результирующая кривая неотрицательна на всем интервале аппроксимации.

График полученного сплайна строится ОМ в виде векторного графа. Также выводится таблица, содержащая значения x, второй производной и квадратного корня дисперсии.

```
program ex2
 use spline_fitting_int
 use show_int
 use rand_int
 use norm_int
 implicit none
 integer :: i, icurv
 integer, parameter :: nbkptin = 13, nord = 4, ndegree = nord - 1,          &
      nbkpt = nbkptin + 2 * ndegree, ndata = 21, ncoeff = nbkpt - nord
 real(kind(1e0)), parameter :: zero = 0e0, one = 1e0, half = 5e-1
 real(kind(1e0)), parameter :: range = 4.0, ratio = 0.2, tol = ratio * half
 real(kind(1e0)), parameter :: delta_x = range / (ndata - 1),               &
      delta_b = range / (nbkptin - 1)
 real(kind(1e0)), target :: xdata(ndata), ydata(ndata), ynoise(ndata),      &
      sddata(ndata), spline_data (3, ndata), bkpt(nbkpt),                   &
      derivat1(ndata), derivat2(ndata), &
      coeff(ncoeff), root_variance(ndata), diff
 real(kind(1e0)), allocatable :: xyvalues(:, :)
 !dec$attributes array_visualizer :: xyvalues
 real(kind(1e0)), dimension(ncoeff,ncoeff) :: sigma_squared
 real(kind(1e0)), pointer :: pointer_bkpt(:)
```

```
type(s_spline_knots) break_points
type(s_spline_constraints) constraints(nbkptin + 2)
allocate(xyvalues(2, ndata))
xdata = (/ ((i - 1) * delta_x, i = 1, ndata) /)
ydata = exp(-half * xdata**2)
xyvalues(1, :) = xdata
ynoise = ratio * ydata * (rand(ynoise) - half)
! Вектор, используемый для проверки результата
ydata = ydata + ynoise
sddata = ynoise
spline_data(1, :) = xdata
spline_data(2, :) = ydata
spline_data(3,:) = sddata
! Задаем узлы сплайна
bkpt = (/ ((i - nord) * delta_b, i = 1, nbkpt) /)
! Присоединяем ссылку к сформированным выше данным (вектору bkpt)
! и задаем степень и узлы сплайна
pointer_bkpt => bkpt
break_points = s_spline_knots(ndegree, pointer_bkpt)
icurv = int(one / delta_b) + 1
! Задаем ограничения, обеспечивающие выпуклость функции вверх при 0 ≤ x ≤ 1
do i = 1, icurv - 1
 constraints(i) = spline_constraints                                    &
      (derivative = 2, point = bkpt(i + ndegree), type = '<=', value = zero)
end do
! Обеспечим равенство нулю второй производной при x = 1
constraints(icurv) = spline_constraints&
      (derivative = 2, point = bkpt(icurv + ndegree), type = '==', value = zero)
! Задаем ограничения, обеспечивающие выпуклость функции вниз при 1 <x ≤ 4
do i = icurv + 1, nbkptin
 constraints(i) = spline_constraints                                    &
      (derivative = 2, point = bkpt(i + ndegree), type = '>=', value = zero)
end do
! Обеспечим равенство первой производной при x = 0
! и неотрицательность кривой на всем интервале аппроксимации
constraints(nbkptin + 1) = spline_constraints                           &
      (derivative = 1, point = bkpt(nord), type = '==', value = zero)
constraints(nbkptin + 2) = spline_constraints                           &
      (derivative = 0, point = bkpt(nbkptin + ndegree), type = '>=', value = zero)
coeff = spline_fitting(data = spline_data, knots = break_points,        &
      constraints = constraints, covariance = sigma_squared)
! Оценка сплайна, квадратного корня дисперсии, первой и второй производных
xyvalues(2, :) = spline_values(0, xdata, break_points, coeff)
```

```
root_variance = spline_values(0, xdata, break_points,                    &
      coeff, covariance = sigma_squared)
derivat1 = spline_values(1, xdata, break_points, coeff)
derivat2 = spline_values(2, xdata, break_points, coeff)
! Вывод таблицы результатов
call show(reshape((/ xdata, derivat2, root_variance /), (/ ndata, 3 /)),  &
      "The x values, 2-nd derivatives, and square root of variance")
! Оценка точности и проверка выполнения заданных ограничений
diff = norm(xyvalues(2, :) - ydata) / norm(ydata)
if(all(xyvalues(2, :) > zero) .and. all(derivat1 < epsilon(zero)).and. diff <= tol) then
 write(*, *) 'Example 2 for SPLINE_FITTING is correct.'
end if
! Для вывода сплайна используем режим векторного графа OM
call vGraph(xyvalues, ndata)          ! Сплайн приведен на рис. 2.2
deallocate(xyvalues)                  ! Текст подпрограммы vGraph см. выше
end program ex2
```

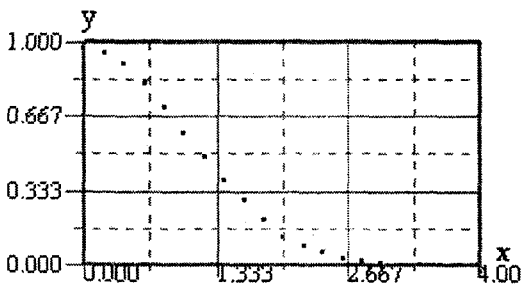


Рис. 2.2. Сглаживание функции $g(x) = e^{-x^2/2}(1 + noise)$ *кубическим сплайном*

Результат (для экономии места выводятся лишь 5 строк таблицы результатов):

The x values, 2-nd derivatives, and square root of variance

	Col		
Row	1	2	3
1	0.000E+00	-5.580E-01	4.877E-03
2	2.000E-01	-9.355E-01	2.339E-03
3	4.000E-01	-1.020E+00	3.340E-03
4	6.000E-01	-5.185E-01	2.511E-03
5	8.000E-01	-2.108E-01	2.528E-03
...	...	...	...

Пример 3. Сплайновая модель датчика почти нормальных случайных чисел.

Функция

$$g(x)=\begin{cases} e^{-x^2/2}, & -1<x<1; \\ 0, & |x|\geq 1 \end{cases}$$

задает почти нормальное распределение вероятности. Функция подобна стандартному нормальному распределению со специальным выбором среднего значения и дисперсии и отличается от него тем, что определена на конечном интервале. В приводимом ниже алгоритме $g(x)$ представляется в виде интерполяционного кубического сплайна $f(x)$. Интегральное распределение получается в результате оценки функции

$$q(x)=\int_{-1}^{x} f(t)dt$$

квадратурными формулами второго порядка Гаусса - Лежандра. Формулы используются на каждом куске $f(t)$, что обеспечивает вычисление $q(x)$ с небольшими затратами. После нормализации кубического сплайна, такой, что $q(1) = 1$, можно генерировать случайные числа согласно распределению $f(t) \equiv q(x)$. Значения x получаются в результате решения уравнения

$$q(x) = u, -1 < x < 1,$$

где u - выборка равномерно распределенных случайных чисел. Уравнение решается методом Ньютона для вектора неизвестных. Рассматриваемый метод генерации случайных чисел на базе равномерного распределения иллюстрируется равенством

$$\frac{d}{dx}(q(x)-u)=f(x), -1<x<1.$$

```
program ex3
 use spline_fitting_int
 use linear_operators
 use Numerical_Libraries
 implicit none
! Используются сплайны для генерации случайных почти нормальных чисел
! Функция нормального распределения задана на интервале (-1, +1)
! Вне интервала она равна нулю
! Дисперсия функции распределения равна 0.5
 integer i, niterat
```

```
 integer, parameter :: iweight = 1, nfix = 0, nord = 4, ndata = 50
 integer, parameter :: nquad = (nord + 1) / 2, ndegree = nord - 1
 integer, parameter :: nbkpt = ndata + 2 * ndegree, ncoeff = nbkpt - nord
 integer, parameter :: last = nbkpt - ndegree, n_samples = 1000
 integer, parameter :: limit = 10
 real(kind(1e0)), dimension(n_samples) :: fn, rn, x, alpha_x, beta_x
 integer left_of(n_samples)
 real(kind(1e0)), parameter :: one = 1e0, half = 5e-1, zero = 0e0, two = 2e0
 real(kind(1e0)), parameter :: delta_x = two / (ndata - 1)
 real(kind(1e0)), parameter :: qalpha = zero, qbeta = zero, domain = two
 real(kind(1e0)) qx(nquad), qxi(nquad), qw(nquad), qxfix(nquad)
 real(kind(1e0)) alpha_, beta_, quad(0:ndata - 1)
 real(kind(1e0)), target :: xdata(ndata), ydata(ndata), coeff(ncoeff),                &
       spline_data(3, ndata), bkpt(nbkpt)
real(kind(1e0)), pointer :: pointer_bkpt(:)
 type(s_spline_knots) break_points
 type(s_spline_constraints) constraints(2)
 ! xyvalues - промежуточный массив, передаваемый отображателю массивов
 real(kind(1e0)), allocatable :: xyvalues(:, :)
 ! Аппроксимируем функцию плотности вероятности сплайнами
 xdata = (/ (-one + (i - 1) * delta_x, i = 1, ndata) /)
 ydata = exp(-half * xdata**2)
 spline_data(1, :) = xdata
 spline_data(2, :) = ydata
 spline_data(3, :) = one
 bkpt = (/ (-one + (i - nord) * delta_x, i = 1, nbkpt) /)
 ! Задаем степень сплайна и его узлы
 pointer_bkpt => bkpt
 break_points = s_spline_knots(ndegree, pointer_bkpt)
 ! Задаем типичные для интерполяционного кубического сплайна
 ! ограничения на производную
 constraints(1) = spline_constraints(derivative = 2, point = bkpt(nord),          &
       type = '==', value = (-one + xdata(1)**2) * ydata(1))
 constraints(2) = spline_constraints(derivative = 2, point = bkpt(last),          &
       type = '==', value = (-one + xdata(ndata)**2) * ydata(ndata))
 ! Вычисляем коэффициенты сплайна
 coeff = spline_fitting(data = spline_data,                                       &
       type = '==',knots = break_points, constraints = constraints)
 ! Выводим рисунок сплайна. Используем режим векторного графа OM
 allocate(xyvalues(2, ndata))
 xyvalues(1, :) = xdata
 xyvalues(2, :) = spline_values(0, xdata, break_points, coeff)
 ! Аппроксимация функции плотности вероятности приведена на рис. 2.3
```

```
call vGraph(xyvalues, ndata)
deallocate(xyvalues)                    ! Текст подпрограммы vGraph см. выше
! Вычисляем оценки точек qx и веса qw, употребляемые
! в квадратурных формулах Гаусса - Лежандра
call gqrul(nquad, iweight, qalpha, qbeta, nfix, qxfix, qx, qw)
! Интегрируем функцию распределения, применяя квадратуры Гаусса - Лежандра
quad(0) = zero
do i = 1, ndata-1
 alpha_ = (bkpt(nord + i) - bkpt(ndegree + i)) * half
 beta_ = (bkpt(nord + i) + bkpt(ndegree + i)) * half
 ! Нормализованные абсциссы приводятся
 ! к интервалам определения кусков сплайна
 ! Каждый кусок интегрируется и накапливается - добавляется -
 ! к прежнему результату
 qxi = alpha_ * qx + beta_
 quad(i) = sum(qw * spline_values(0, qxi, break_points, coeff)) * alpha_ + quad(i - 1)
end do
! Нормализуем коэффициенты и интегралы по отдельным кускам так,
! чтобы общий интеграл был равен единице
coeff = coeff / quad(ndata - 1)
quad = quad / quad(ndata - 1)
rn = rand(rn)
x = zero; niterat = 0
solve_equation: do
 ! Находим интервалы размещения значений x
 left_of = ndegree; i = ndegree
 do
  i = i + 1; if(i >= last) exit
  where(x >= bkpt(i)) left_of = left_of + 1
 end do
 ! Используем метод Ньютона для решения нелинейного уравнения
 ! накопленная функция распределения - случайное число = 0
 alpha_x = (x - bkpt(left_of)) * half
 beta_x = (x + bkpt(left_of)) * half
 fn = quad(left_of - nord) - rn
 do i = 1, nquad
  fn = fn + qw(i) * spline_values(0, alpha_x * qx(i) + beta_x,                    &
       break_points, coeff) * alpha_x
 end do
 ! Выполняем ньютоновский шаг
 x = x - fn / spline_values(0, x, break_points, coeff)
 niterat = niterat + 1
 ! Ограничиваем значения так, чтобы они находились внутри интервала
```

```
 where(x <= -one .or. x >= one) x = zero
 if(norm(fn, 1) <= sqrt(epsilon(one)) * norm(x, 1)) exit solve_equation
end do solve_equation
! Проверяем сходимость метода Ньютона
if(niterat <= limit) print *, 'Example 3 for SPLINE_FITTING is correct.'
end program ex3
```

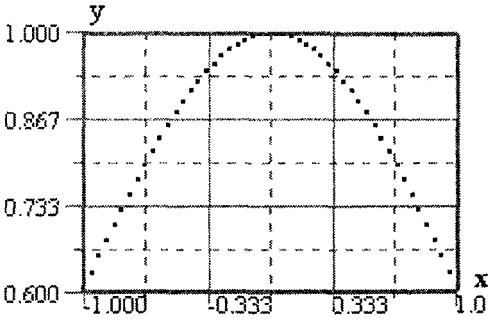


Рис. 2.3. Аппроксимация функции плотности вероятности

Пример 4. Представление периодической кривой.

Формируемый сплайн аппроксимирует прямоугольник. Сплайн представляется в виде параметрической функции ($x(t)$, $y(t)$). Степень сплайна равна единице, т. е. выполняется кусочно-линейная аппроксимация прямоугольника. Задаваемые ограничения делают сплайн-функцию периодической. Некоторые узлы сплайна совпадают с углами прямоугольника, в которых производная претерпевает разрыв. Значения параметра t задаются на сторонах прямоугольника, что упрощает задачу аппроксимации. Пример иллюстрирует метод представления ребра двумерной области, ограниченной периодической кривой.

```
program ex4
 use spline_fitting_int
 use norm_int
 implicit none
 ! Кусочно-линейный сплайн используется для кусочно-линейной
 ! аппроксимации прямоугольника
 integer i, j
 integer, parameter :: nbkpt = 9, nord = 2, ndegree = nord - 1,                    &
       ncoeff = nbkpt - nord, ndata = 7, ngrid = 100, nvalues = (ndata - 1) * ngrid
 real(kind(1e0)), parameter :: zero = 0e0, one = 1e0
 real(kind(1e0)), parameter :: delta_t = one, delta_b = one, delta_v = 0.01
 real(kind(1e0)) delta_x, delta_y
```

```
! Дополнительные координаты на ребрах прямоугольника
real(kind(1e0)), dimension(ndata) :: sddata = one,                    &
      xdata = (/ 0.0, 1.0, 2.0, 2.0, 1.0, 0.0, 0.0 /),                &
      ydata = (/ 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0 /)
real(kind(1e0)) tdata(ndata), xspline_data(3, ndata),                 &
      yspline_data(3, ndata), tvalues(nvalues),                       &
      xvalues(nvalues), yvalues(nvalues), xcoeff(ncoeff),             &
      ycoeff(ncoeff), xcheck(nvalues), ycheck(nvalues), diff
! xyvalues - промежуточный массив, передаваемый отображателю массивов
real(kind(1e0)), allocatable :: xyvalues(:, :)
real(kind(1e0)), target :: bkpt(nbkpt)
real(kind(1e0)), pointer :: pointer_bkpt(:)
type(s_spline_knots) break_points
type(s_spline_constraints) constraints(1)
tdata = (/ ((i - 1) * delta_t, i = 1, ndata) /)
xspline_data(1, :) = tdata; yspline_data(1, :) = tdata
xspline_data(2, :) = xdata; yspline_data(2, :) = ydata
xspline_data(3, :) = sddata; yspline_data(3, :) = sddata
! Задаем узлы сплайна
bkpt(nord:nbkpt - ndegree) = (/ ((i - nord) * delta_b, i = nord, nbkpt - ndegree) /)
! Устраняем неверные значения узлов сплайна
bkpt(1:ndegree) = bkpt(nord)
bkpt(nbkpt - ndegree + 1:nbkpt) = bkpt(nbkpt - ndegree)
! Присоединяем ссылку к сформированным выше данным (вектору bkpt)
! и задаем степень и контрольные точки сплайна
pointer_bkpt => bkpt
break_points = s_spline_knots(ndegree, pointer_bkpt)
! Делаем две параметрические кривые периодическими
constraints(1) = spline_constraints(derivative = 0, point = bkpt(nord),  &
      type = '.=.', value = bkpt(nbkpt - ndegree))
xcoeff = spline_fitting(data = xspline_data, knots = break_points,       &
      constraints = constraints)
ycoeff = spline_fitting(data = yspline_data, knots = break_points,       &
      constraints = constraints)
! Вычисляем значения найденного сплайна в узлах, расположенных
! на сторонах прямоугольника, и для проверки сравниваем их
! с точками на сторонах прямоугольника
tvalues= (/ ((i - 1) * delta_v, i = 1, nvalues) /)
xvalues = spline_values(0, tvalues, break_points, xcoeff)
yvalues = spline_values(0, tvalues, break_points, ycoeff)
do i=1, nvalues
 j = (i - 1) / ngrid + 1
 delta_x = (xdata(j + 1) - xdata(j)) / ngrid
```

```
  delta_y = (ydata(j + 1) - ydata(j)) / ngrid
  xcheck(i) = xdata(j) + mod(i + ngrid - 1, ngrid) * delta_x
  ycheck(i) = ydata(j) + mod(i + ngrid - 1, ngrid) * delta_y
 end do
 diff = norm(xvalues - xcheck, 1) / norm(xcheck, 1) +                    &
        norm(yvalues - ycheck, 1) / norm(ycheck, 1)
 if(diff <= sqrt(epsilon(one))) print *, 'Example 4 for SPLINE_FITTING is correct.'
 ! Для вывода сплайна используем режим векторного графа OM
 allocate(xyvalues(2, nvalues))
 xyvalues(1, :) = xvalues
 xyvalues(2, :) = yvalues
 call vGraph(xyvalues, nvalues)     ! Сплайн-прямоугольник приведен на рис. 2.4
 deallocate(xyvalues)               ! Текст подпрограммы vGraph см. выше
end program ex4
```

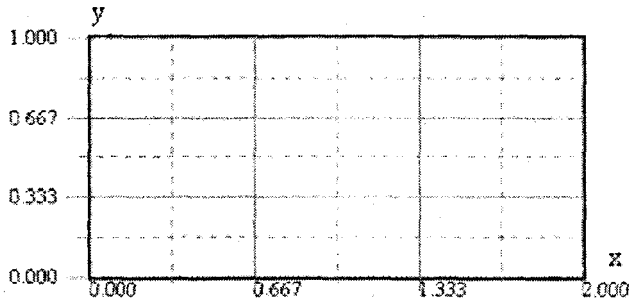


Рис. 2.4. Представление прямоугольника в виде кусочно-линейного сплайна

2.3. ОПИСАНИЕ ФУНКЦИЙ, УПОТРЕБЛЯЕМЫХ С ПРОСТРАНСТВЕННЫМИ СПЛАЙНАМИ

2.3.1. ФУНКЦИЯ SURFACE_CONSTRAINTS

Возвращает массив производного типа ?*_surface_constraints*. Синтаксис вызова функции с ключевыми словами для узла с номером *j*:

```
?_surface_constraints(j) = SURFACE_CONSTRAINTS                          &
      ([derivative = derivative_index(1:2),]                            &
      point = where_applied(1:2), [value = value_applied],              &
      type = constraint_indicator,                                      &
      [periodic_point = periodic_point(1:2)])
```

Указанные в квадратных скобках параметры являются необязательными. Нельзя задавать параметры *value* = или *periodic_point* = одновременно.

Все параметры функции, как обязательные, так и необязательные, являются *входными*. Тип числовых параметров - REAL(4) или REAL(8).

Обязательные параметры:

point = *where_applied* - точка, в которой должны быть выполнены задаваемые ограничения. Точка принадлежит к области определения сплайна. Каждая точка задается своими *x*- и *y*-координатами.

type = *constraint_indicator* - вид ограничения для сплайна или его производной. Описание ограничений приведено в разд. 2.2.1.

Необязательные параметры:

derivative = *derivative_index*(1:2) - номер производной формируемого сплайна, которая должна удовлетворять задаваемым ограничениям. Массив (/ 0, 0 /) соответствует функции, массив (/ 1, 0 /) - первой частной производной по *x* и т. д. Если параметр опущен, то *derivative* = 0, т. е. ограничению должна отвечать сплайн-функция.

periodic_point = *periodic_point*(1:2) - автоматическое определение второй пары независимых переменных для периодических ограничений.

2.3.2. ФУНКЦИЯ SURFACE_VALUES

Возвращает в качестве результата массив ранга 2, отвечающий двум входным векторам. Использует, когда вычисляется квадратный корень дисперсии, в качестве необязательного параметра матрицу ковариаций.

Результатом будет скаляр, если на входе функции - скаляр.

Функция имеет следующий вызов:

zvalues = SURFACE_VALUES(*derivative* = *derivative*, &
variablesx = *variablesx*, *variablesy* = *variablesy*, &
knotsx = *knotsx*, *knotsy* = *knotsy*, &
coeffs = *c*, [*covariance* = *G*], [*iopt* = *iopt*])

Все параметры функции являются *входными*. Тип числовых параметров - REAL(4) или REAL(8). Массив - результат имеет форму (SIZE(*variablesx*), SIZE(*variablesy*)).

Обязательные параметры:

derivative = *derivative*(1:2) - номер вычисляемой частной производной. Не может быть меньше нуля. Для оценки сплайн-функции используется массив (/ 0, 0 /).

variablesx = *variablesx* и *variablesy* = *variablesy* - соответственно значения независимых переменных по осям *x* и *y*, при которых выполняется оценка

сплайна или его производных. В качестве *variablesx* (*variablesy*) употребляется либо вектор (массив ранга 1), либо скаляр.

knotsx = *knotsx* и *knotsy* = *knotsy* - переменные производного типа ?_*spline_knots*, используемые для получения массива *coeffs*(:, :) функцией SURFACE_FITTING. Содержат степень сплайна, число его узлов и сами узлы по осям *x* и *y*.

coeffs = *c* - коэффициенты в представлении сплайн-функции

$$f(x,y)=\sum_{j=1}^{n}\sum_{i=1}^{m}c_{ij}B_i(y)B_j(x)\,. \qquad (2.4)$$

Коэффициенты возвращаются в результате вызова

```
c = surface_fitting(...)
```

порядок выполнения которого приводится ниже. Причем величины *m* = SIZE(*c*, 1) и *n* = SIZE(*c*, 2) удовлетворяют соответственно соотношениям

$$n - 1 + spline\_degree = \mathrm{SIZE}(?\_knotsx) \qquad (2.5)$$

и

$$m - 1 + spline\_degree = \mathrm{SIZE}(?\_knotsy), \qquad (2.6)$$

в которых *spline_degree*, ?_*knotsx* и ?_*knotsy* - компоненты параметров *knotsx* и *knotsy*. Одно и то же значение *spline_degree* должно быть использовано и для *knotsx* и для *knotsy*.

Необязательные параметры:

covariance = *G* - параметр, если присутствует, равен квадратному корню дисперсии функции

$$e(x,y)=\sqrt{b(x,y)^T\,Gb(x,y)}\,,$$

где

$$b(x,y)=(B_1(x)\,B_1(y),\ ...,\ B_n(x)B_1(y),\ ...)^T,$$

а *G* - матрица ковариации, связанная с коэффициентами сплайна

$$c=(c_{11},\ ...,\ c_{n1},\ ...)^T.$$

Параметр *G* является необязательным. Он возвращается обсуждаемой ниже функцией SURFACE_FITTING. Когда вычисляется квадратный корень функции дисперсии, параметры *derivative* и *c* не используются.

iopt = *iopt* - необязательный параметр производного типа ?_*options*. В настоящей версии не используется.

2.3.3. ФУНКЦИЯ SURFACE_FITTING

Возвращает взвешенное приближение функции B-сплайном (2.4), найденное по дискретным двумерным данным с использованием метода наименьших квадратов. Ограничения на сплайн и его производные могут быть опущены. После выполнения приближения можно, применив SURFACE_VALUES, оценить сплайн-функцию, ее производные или квадратный корень дисперсии. Функция имеет следующий вызов:

```
c = SURFACE_FITTING(data = data(1:4, :),                    &
      knotsx = knotsx, knotsy = knotsy,                     &
      constraints = surface_constraints,                    &
      [covariance = G], [iopt = iopt])
```

Параметры функции SURFACE_FITTING:

Входные: knotsx, knotsy, constraints.

Входные/выходные: data, iopt.

Выходной: covariance.

Функция возвращает массив коэффициентов сплайна, размер которого вычисляется по формулам (2.5) и (2.6).

Обязательные параметры функции SURFACE_FITTING:

data = data(1:3, :) - перенимающий форму массив, содержащий данные в следующем порядке: *data*(1, *i*) = x_i, *data*(2, *i*) = y_i, *data*(3, *i*) = z_i и *data*(4, *i*) = σ_i, *i* = 1, ..., *ndata*. Если дисперсии неизвестны, но пропорциональны неизвестной величине, то можно задать *data*(4, *i*) = 1, *i* = 1, ..., *ndata*. Параметр имеет тип REAL(4) или REAL(8).

knotsx = knotsx и *knotsy = knotsy* - параметры производного типа ?*_spline_zknots*, определяющие степень сплайна и его узлы в области приближения соответственно по осям *x* и *y*.

Необязательные параметры функции SURFACE_FITTING:

constraints = surface_constraints - вектор производного типа ?*_spline_constraints*, задающий ограничения, которым сплайн должен удовлетворять. Тип параметра совпадает с типом массива *data*.

covariance = G - перенимающий форму массив ранга 2, имеющий ту же точность, что и массив *data*. Содержит на выходе матрицу ковариаций коэффициентов. Используется для вычисления квадратного корня дисперсии.

iopt = iopt(:) - вектор производного типа. Используется для пересылки необязательных данных функции SURFACE_FITTING. Работает с приведенными в табл. 2.2 опциями.

Таблица 2.2. Опции, употребляемые с параметром iopt

Опция	*Значение опции*
surface_fitting_smallness	1
surface_fitting_flatness	2
surface_fitting_tol_equal	3
surface_fitting_tol_least	4
surface_fitting_residuals	5
surface_fitting_print	6
surface_fitting_thinness	7

iopt(*io*) = ?_options(*surface_fitting_smallnes*, ?_*value*) - устанавливает значение квадратного корня параметра регуляризации, на который умножается двойной интеграл неизвестной функции. Параметр ?_*value* замещает значение, заданное по умолчанию. По умолчанию ?_*value* = 0.

iopt(*io*) = ?_options(*surface_fitting_flatness*, ?_*value*) - устанавливает значение квадратного корня параметра регуляризации, на который умножается двойной интеграл частной производной неизвестной функции. Параметр ?_*value* замещает значение, заданное по умолчанию. По умолчанию ?_*value* = SQRT(EPSILON(?_*value*)) * *size*, где *size* = SUM(ABS(*data*(3, :)/ *data*(4, :))/(*ndata* + 1).

iopt(*io*) = ?_options(*spline_fitting_tol_equal*, ?_*value*) - устанавливает значение для определения ситуации, когда ограничения равенства имеют дефицит ранга. Значение по умолчанию ?_*value* = 10^{-4}.

iopt(*io*) = ?_options(*spline_fitting_tol_least*, ?_*value*) - устанавливает значение для определения ситуации, когда уравнения в модели наименьших квадратов имеют дефицит ранга. Значение по умолчанию ?_*value* = 10^{-4}.

iopt(*io*) = ?_options(*surface_fitting_residuals*, *dummy*) - возвращает в *data*(4, :) невязки *residuals* = *surface* - *data*. Данные возвращаются в порядке обработки элементов, т. е. слева направо по оси *x* и снизу вверх по оси *y*. По умолчанию невязки не вычисляются и заданные на входе значения *data*(1:4, :) сохраняются.

iopt(*io*) = ?_options(*surface_fitting_print*, *dummy*) - обеспечивает вывод *x*- и *y*-координат узлов сплайна и рассчитанных данных в порядке их обработки. По умолчанию данные не выводятся.

iopt(*io*) = ?_options(*surface_fitting_thinness*, ?_*value*) - устанавливает значение квадратного корня параметра регуляризации, на который умножается двойной интеграл второй частной производной неизвестной функции. Параметр ?_*value* замещает значение, заданное по умолчанию. По умолчанию ?_*value* = 10^{-3} * *size*, где *size* = SUM(ABS(*data*(3, :) / *data*(4, :)) / (*ndata* + 1).

Описание:

Коэффициенты сплайна получаются в результате решения по методу наименьших квадратов системы взвешенных линейных алгебраических уравнений с ограничениями равенства и неравенства. При отсутствии ограничений результат вычисляется ленточным решателем задачи наименьших квадратов. Детали см. в [34].

Коэффициенты *c* получаются в результате решения проблемы наименьших квадратов с линейными ограничениями равенства и неравенства. Результатом решения системы являются коэффициенты сплайн-функции (2.4).

При отсутствии ограничений решение находится рассмотренным в [34] решателем задачи наименьших квадратов.

Замечание. Сообщения о завершающих ошибках находятся в файле messages.gls. Они имеют номера 1151-1152, 1161-1162, 1370-1393.

Пример 1. Функция

$$g(x, y) = \exp(-x^2 - y^2)$$

приближается по методу наименьших квадратов тензорным произведением кубических сплайнов в квадрате $[0, 2]\times[0, 2]$.

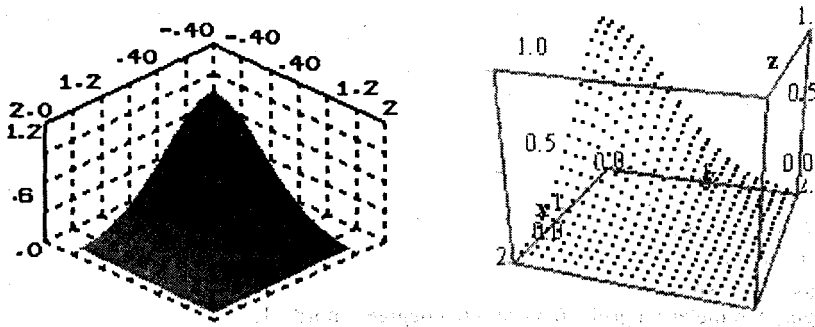
В программе задаются *ndata* случайных пар значений независимых переменных. Дисперсия зависимой переменной устанавливается равной единице. Сетка узлов внутри квадрата равномерна как по измерению *x*, так и по измерению *y*. После вычисления коэффициентов выполняется проверка, насколько точно B-сплайн аппроксимирует поверхность $g(x, y)$. Вывод B-сплайна осуществляется ОМ, функционирующим в режиме векторного графа.

```
program ex_suf1
 use surface_fitting_int
 use rand_int
 use norm_int
 implicit none
! Порядок сплайна nord = 4, а число ячеек сетки равно (ngrid - 1)**2
! Внутри квадрата задаются для построения сплайна ndata = 2000 значений
integer :: i, kb
integer, parameter :: ngrid = 9, nord = 4, ndegree = nord - 1, ndata = 2000,        &
    nbkpt = ngrid + 2 * ndegree, nvalues = 100, kv = 5, nvalkv = nvalues / kv
real(kind(1d0)), parameter :: zero = 0d0, one = 1d0, two = 2d0
real(kind(1d0)), parameter :: tolerance = 1d-3
real(kind(1d0)), target :: spline_data(4, ndata), bkpt(nbkpt),                    &
    coeff(ngrid + ndegree - 1, ngrid + ndegree - 1), delta, sizev,                &
    x(nvalues), y(nvalues), values(nvalues, nvalues)
```

```
real(kind(1d0)), pointer :: pointer_bkpt(:)
type(d_spline_knots) knotsx, knotsy
! Bspline - массив, введенный для графического отображения B-сплайна
real(4) :: Bspline(3, nvalkv * nvalkv)
! Генерируем случайным образом (x, y)-пары и оцениваем
! заданную функцию в этих точках
spline_data(1:2, :) = two * rand(spline_data(1:2, :))
spline_data(3, :) = exp(-sum(spline_data(1:2, :)**2, dim = 1))
spline_data(4, :) = one
delta = two / (ngrid - 1)                    ! Формируем узлы аппроксимации
bkpt(1:ndegree) = zero
bkpt(nbkpt - ndegree + 1:nbkpt) = two
bkpt(nord:nbkpt - ndegree) = (/ (i * delta, i = 0, ngrid - 1) /)
! Находим knotsx и knotsy - параметры функции SURFACE_FITTING
pointer_bkpt => bkpt
knotsx = d_spline_knots(ndegree, pointer_bkpt)
knotsy = knotsx
! Выполняем приближение поверхности сплайном; получаем его коэффициенты
coeff = surface_fitting(spline_data, knotsx, knotsy)
! Оцениваем невязки residual = spline - function в точках сетки внутри квадрата
delta = two / (nvalues + 1)
x = (/ (i * delta, i = 1, nvalues) /); y = x
! Оцениваем B-сплайн
values = surface_values((/ 0, 0 /), x, y, knotsx, knotsy, coeff)
! Графическое отображение результата приведено на рис. 2.5
kb = 1
do i = 1, nvalues, kv
 Bspline(1, kb:kb + nvalkv - 1) = x(i)
 Bspline(2, kb:kb + nvalkv - 1) = y(1:nvalues:kv)
 Bspline(3, kb:kb + nvalkv - 1) = values(i, 1:nvalues:kv)
 kb = kb + nvalkv
end do                                       ! Текст подпрограммы vGraph2 см. ниже
! Вызов OM
call vGraph2(Bspline, nvalkv * nvalkv)
values = values - exp(-spread(x**2, 1, nvalues) - spread(y**2, 2, nvalues))
! Вычисляем среднеквадратичную ошибку
sizev = norm(pack(values, (values == values))) / nvalues
if(sizev <= tolerance) write(*, *) 'Example 1 for surface_fitting is correct.'
end program ex_suf1

subroutine vGraph2(fun, nvalues)        ! Выводит массив как векторный граф OM
 use avdef
 use avviewer
```

```
use dflib
integer(4) :: nvalues                    ! Подпрограмма vGraph2 выводит в режиме
real(4) :: fun(3, nvalues)               ! векторного графа трехмерный образ,
integer(4) hv, status, nError            ! в то время как подпрограмма vGraph
character(1) :: key                      ! формирует плоское изображение
character(av_max_label_len) :: xLabel = 'x', yLabel = 'y', zLabel = 'z'
call faglStartWatch(fun, status)         ! Сообщаем ОМ имя отображаемого массива
print *, "Starting Array Viewer"
! Запуск ОМ с использованием fav-подпрограммы
call favStartViewer(hv, status)
if(status /= 0) then
 call favGetErrorNo(hv, nError, status)
 if(nError /= 0) then
  print *, "Array Viewer reports error ", nError
  stop
 end if
end if
! Передаем ОМ данные подлежащего отображению массива
call favSetArray(hv, fun, status)
! Задаем заголовок экземпляра ОМ
call favSetArrayName(hv, "Bspline(x, y)", status)
! Отображаем массив в виде векторного графа
call favSetGraphType(hv, VectorGraph, status)
! Задаем режим вывода заданных пользователем имен осей координат
call favSetUseAxisLabel(hv, x_axis, 1, status)
call favSetUseAxisLabel(hv, y_axis, 1, status)
call favSetUseAxisLabel(hv, z_axis, 1, status)
! Новые (вместо dim1, dim2 и z) имена x- и y-осей координат. Длина переменной,
! задающей имя оси, равна AV_MAX_LABEL_LEN
call favSetAxisLabel(hv, x_axis, xLabel, status)
call favSetAxisLabel(hv, y_axis, yLabel, status)
call favSetAxisLabel(hv, z_axis, zLabel, status)
! Устанавливаем режим явного задания разметок координатных осей
call favSetAxisAutoDetail(hv, 0, status)
! Число больших разметок на координатных осях
call favSetNumMajorTickmarks(hv, x_axis, 3, status)
call favSetNumMajorTickmarks(hv, y_axis, 3, status)
call favSetNumMajorTickmarks(hv, z_axis, 3, status)
! Показываем ОМ на экране
call favShowWindow(hv, av_true, status)
print *, "Press any key to close down the viewer"
key = getcharqq( )
call favEndViewer(hv, status)            ! Закрываем ОМ
```

```
call faglEndWatch(fun, status)          ! Освобождаем ресурсы
end subroutine vGraph2
```



а *б*

Рис. 2.5. Функция g(x, y) = exp(-x² - y²):
а - тоновая модель; б - приближение B-сплайном

Пример 2. Выполняется сглаживание сферы, координаты которой представлены в параметрическом виде. Параметрическое представление точки сферы (x, y, z) радиуса $a > 0$ выражается в сферических координатах следующим образом:

$$x(u, v) = a\cos(u)\cos(v), -\pi \le 2u \le \pi;$$

$$y(u, v) = a\cos(u)\sin(v), -\pi \le v \le \pi;$$

$$z(u, v) = a\sin(u).$$

Параметры u и v измеряются в радианах и обозначают соответственно широту (угол между экватором и параллелью) и долготу (угол между нулевым меридианом и меридианом, проходящим через точку на сфере). В программе, поскольку широта $-\pi \le u \le \pi$, сфера обходится дважды. При этом задача сглаживания решается для каждой из координат - функций $x(u, v)$, $y(u, v)$ и $z(u, v)$. На переменные u и v накладываются периодические ограничения. Заметим, что объем вычислений, необходимых для поиска сплайна, можно существенно снизить, выполняя сглаживание только z-координат точек сферы. Однако для демонстрации механизма представления более сложных, чем сфера, поверхностей такого упрощения не выполняется. При оценке поверхности широта изменяется от южного полюса к северному. Получаемая сфера приблизительно соответствует равенству

$$x^2 + y^2 + z^2 = a^2.$$

Точность приближения оценивается по невязкам, вычисляемым на прямоугольной сетке для каждой пары широты и долготы. Для демонстрации использования опций все 3 параметра регуляризации устанавливаются равными нулю, допуск задается меньшим значения по умолчанию, а невязки вычисляются для каждой из функций.

```
program ex_suf2
 use surface_fitting_int
 use rand_int
 use norm_int
 use Numerical_Libraries
 implicit none
 integer :: i, j, kb
 integer, parameter :: ngrid = 6, nord = 6, ndegree = nord - 1,                    &
      nbkpt = ngrid + 2 * ndegree, ndata = 1000, nvalues = 50, nopt = 5
 real(kind(1d0)), parameter :: zero = 0d0, one = 1d0, two = 2d0
 real(kind(1d0)), parameter :: tolerance = 1d-2
 real(kind(1d0)), target :: spline_data (4, ndata, 3), bkpt(nbkpt),                &
      coeff(ngrid + ndegree - 1, ngrid + ndegree - 1, 3), delta, sizev,            &
      pi, a, x(nvalues), y(nvalues), values(nvalues, nvalues), data(4, ndata),     &
      bx(nvalues, nvalues), by(nvalues, nvalues)
 real(kind(1d0)), pointer :: pointer_bkpt(:)
 type(d_spline_knots) knotsx, knotsy
 type(d_options) options(nopt)
 ! Массив, введенный для графического отображения B-сплайна
 real(4), allocatable :: Bspline(:, :)
 !dec$attributes array_visualizer :: Bspline
 allocate(Bspline(3, nvalues * nvalues))
 ! Получаем константу π и радиус сферы как случайное число плюс 1.0
 pi = dconst((/ "pi" /)); a = one + rand(a)
 ! Генерируем случайным образом широту и долготу
 spline_data(1:2, :, 1)=pi * (two * rand(spline_data(1:2, :, 1)) - one)
 spline_data(1:2, :, 2)=spline_data(1:2, :, 1)
 spline_data(1:2, :, 3)=spline_data(1:2, :, 1)
 ! Оцениваем x-, y- и z-координаты точки на сфере
 spline_data(3, :, 1)=a * cos(spline_data(1, :, 1)) * cos(spline_data(2, :, 1))
 spline_data(3, :, 2)=a * cos(spline_data(1, :, 2)) * sin(spline_data(2, :, 2))
 spline_data(3, :, 3)=a * sin(spline_data(1, :, 3))
 ! Все значения обладают одинаковой неопределенностью
 spline_data(4, :, :) = one
 delta = two * pi / (ngrid - 1)              ! Задаем узлы задачи сглаживания
 bkpt(1:ndegree) = -pi
 bkpt(nbkpt - ndegree + 1:nbkpt) = pi
```

```
bkpt(nord:nbkpt - ndegree) = (/ (-pi + i * delta, i = 0, ngrid - 1) /)
! Задаем степень B-сплайна
! и вычисляем knotsx и knotsy - параметры подпрограммы SURFACE_FITTING
pointer_bkpt => bkpt
knotsx = d_spline_knots(ndegree, pointer_bkpt); knotsy = knotsx
! Решаем задачу сглаживания для каждой из координат -
! функций x(u, v), y(u, v) и z(u, v)
! Изменяем параметры регуляризации
! Задаем вычисление невязок в каждой точке. Невязки возвращаются в data(4, :)
do j = 1, 3
 data = spline_data(:, :, j)
 options(1) = d_options(surface_fitting_thinness, zero)
 options(2) = d_options(surface_fitting_flatness, zero)
 options(3) = d_options(surface_fitting_smallness, zero)
 options(4) = d_options(surface_fitting_tol_least, 1d-5)
 options(5) = surface_fitting_residuals
 coeff(:, :, j) = surface_fitting(data, knotsx, knotsy, iopt = options)
end do
! Выполняем оценку функции внутри содержащего сетку прямоугольника
! Сумма квадратов координат x^2 + y^2 + z^2 отличается от a**2
! из-за ошибок округления
delta = pi / (nvalues + 1)
x = (/(-pi / two + i * delta, i = 1, nvalues) /); y = two * x
values = zero
do j = 1, 3
 values = values + surface_values((/ 0, 0 /), x, y, knotsx, knotsy, coeff(:, :, j))**2
end do
values = values - a**2
! Вычисляем среднеквадратичную ошибку
sizev = norm(pack(values, (values == values))) / nvalues
if(sizev <= tolerance) write(*, *) "Example 2 for SURFACE_FITTING is correct."
! Графическое отображение результата приведено на рис. 2.6
bx = surface_values((/ 0, 0 /), x, y, knotsx, knotsy, coeff(:, :, 1))
by = surface_values((/ 0, 0 /), x, y, knotsx, knotsy, coeff(:, :, 2))
values = surface_values((/ 0, 0 /), x, y, knotsx, knotsy, coeff(:, :, 3))
kb = 0
do i = 1, nvalues
 do j = 1, nvalues
  kb = kb + 1
  Bspline(1, kb) = bx(i, j)
  Bspline(2, kb) = by(i, j)
  Bspline(3, kb) = values(i, j)
 end do
```

```
end do                                    ! Текст подпрограммы vGraph2 см. выше
! Вызов ОМ
call vGraph2(Bspline, nvalues * nvalues)
deallocate(Bspline)
end program ex_suf2
```

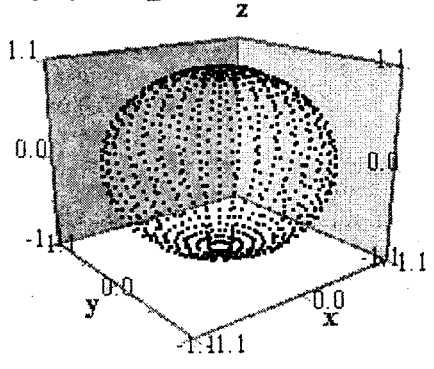


Рис. 2.6. Приближение сферы B-сплайном

Пример 3. Приближается B-сплайном та же, что и в примере 1, поверхность $g(x, y) = \exp(-x^2 - y^2)$. Задача решается в квадрате $[0, 2]\times[0, 2]$. При этом демонстрируется механизм применения дискретных ограничений на форму поверхности. Ограничения таковы, что $g(0, 0) = 1$, $\partial g(0, 0)/\partial x = 0$ и $\partial g(0, 0)/\partial y = 0$. Также вычисляются невязки в узлах сетки и в точке (0, 0).

```
program ex_suf3
 use surface_fitting_int
 use rand_int
 use norm_int
 implicit none
 integer :: i, kb
 integer, parameter :: ngrid = 9, nord = 4, ndegree = nord - 1, ndata = 2000,          &
      nbkpt = ngrid + 2 * ndegree, nvalues = 100, nc = 3, kv = 5, nvalkv = nvalues / kv
 real(kind(1d0)), parameter :: zero = 0d0, one = 1d0, two = 2d0
 real(kind(1d0)), parameter :: tolerance = 1d-3
 real(kind(1d0)), target :: spline_data (4, ndata), bkpt(nbkpt),                       &
      coeff(ngrid + ndegree - 1, ngrid + ndegree - 1), delta, sizev,                   &
      x(nvalues), y(nvalues), values(nvalues, nvalues), f_00, f_x00, f_y00
 real(kind(1d0)), pointer :: pointer_bkpt(:)
 type(d_spline_knots) knotsx, knotsy
 type(d_surface_constraints) c(nc)
 logical pass
 ! Массив, введенный для графического отображения B-сплайна
 real(4), allocatable :: Bspline(:, :)
```

```
!dec$attributes array_visualizer :: Bspline
allocate(Bspline(3, nvalkv * nvalkv))
! Генерируем случайные координаты (x, y) и оцениваем функцию в этих точках
spline_data(1:2, :) = two * rand(spline_data(1:2, :))
spline_data(3, :) = exp(-sum(spline_data(1:2, :)**2, dim = 1))
spline_data(4, :) = one
! Задаем узлы для решения задачи приближения
delta = two / (ngrid - 1)
bkpt(1:ndegree) = zero
bkpt(nbkpt - ndegree + 1:nbkpt) = two
bkpt(nord:nbkpt - ndegree) = (/( i * delta, i = 0, ngrid - 1) /)
! Задаем степень B-сплайна
! и вычисляем knotsx и knotsy - параметры функции SURFACE_FITTING
pointer_bkpt => bkpt
knotsx = d_spline_knots(ndegree, pointer_bkpt)
knotsy = knotsx
! Задаем ограничения g(0, 0) = 1, ∂g(0, 0)/∂x = 0 и ∂g(0, 0)/∂y = 0
c(1) = surface_constraints(point = (/zero, zero/), type = '==', value = one)
c(2) = surface_constraints(derivative = (/ 1, 0 /), point = (/ zero, zero /),          &
      type = '==', value = zero)
c(3) = surface_constraints(derivative = (/ 0, 1 /), point = (/ zero, zero /),          &
      type = '==', value = zero)
! Выполняем приближение и получаем коэффициенты B-сплайна
coeff = surface_fitting(spline_data, knotsx, knotsy, constraints = c)
! Формируем массивы x- и y-координат для оценки и вывода B-сплайна
delta = two / (nvalues + 1)
x = (/ (i * delta, i = 1, nvalues) /); y = x
values = exp(-spread(x**2, 1, nvalues) - spread(y**2, 2, nvalues))
! Графическое отображение результата приведено на рис. 2.7
kb = 1
do i = 1, nvalues, kv
 Bspline(1, kb:kb + nvalkv - 1) = x(i)
 Bspline(2, kb:kb + nvalkv - 1) = y(1:nvalues:kv)
 Bspline(3, kb:kb + nvalkv - 1) = values(i, 1:nvalues:kv)
 kb = kb + nvalkv
end do                                  ! Текст подпрограммы vGraph2 см. выше
! Вызов ОМ
call vGraph2(Bspline, nvalkv * nvalkv)
deallocate(Bspline)
! Оцениваем невязки residual = spline - function в точках квадрата [0, 2]×[0, 2]
values = surface_values((/ 0, 0 /), x, y, knotsx, knotsy, coeff) - values
f_00 = surface_values((/0, 0/), zero, zero, knotsx, knotsy, coeff)
f_x00 = surface_values((/1, 0/), zero, zero, knotsx, knotsy, coeff)
```

```
f_y00 = surface_values((/0, 1/), zero, zero, knotsx, knotsy, coeff)
! Вычисляем среднеквадратичную ошибку
sizev = norm(pack(values, (values == values))) / nvalues
pass = sizev <= tolerance
pass = abs (f_00 - one) <= sqrt(epsilon(one)) .and. pass
pass = f_x00 <= sqrt(epsilon(one)) .and. pass
pass = f_y00 <= sqrt(epsilon(one)) .and. pass
if(pass) write(*, *) 'Example 3 for SURFACE_FITTING is correct.'
end program ex_suf3
```

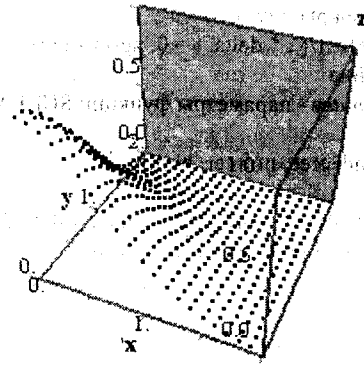


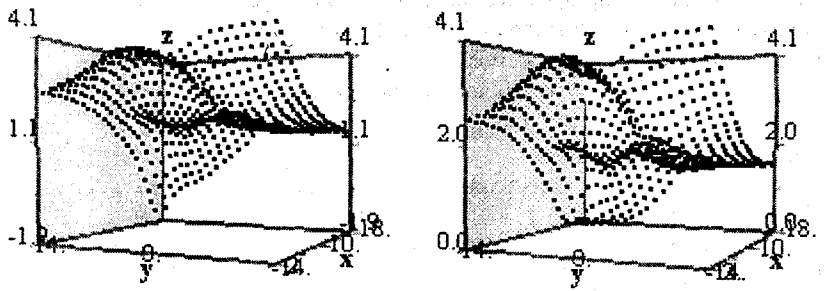
Рис. 2.7. B-сплайн, приближающий поверхность $g(x, y) = exp(-x^2 - y^2)$ в квадрате [0, 2] ×[0, 2] с ограничениями $g(0, 0) = 1$, $\partial g(0, 0)/\partial x = 0$ и $\partial g(0, 0)/\partial y = 0$

Пример 4. Формируется сплайновая поверхность; ограничения таковы, что поверхность является неотрицательной. Тестовые данные взяты из [26]. Они содержат 25 точек с единичной неопределенностью для каждой зависимой переменной. По этим точкам находится по методу наименьших квадратов сглаживающая поверхность. Перед сглаживанием устанавливаются значения рассмотренных в [34] параметров регуляризации - пологости и разреженности. Затем выполняется сглаживание поверхности и оценка результата в узлах сетки. Без ограничений сглаживающая функция имеет отрицательные значения около одного из углов области сглаживания. Чтобы избежать отрицательных значений функции, в программе задаются ограничения, обеспечивающие неотрицательность функции во всех точках прямоугольника, содержащего пары независимых переменных. Применяемый подход к заданию ограничений прост, но эффективен. Его суть в том, что сглаживание выполняется заново с соответствующими ограничениями в тех точках сетки, где ранее найденный B-сплайн отрицателен.

```
program ex_suf4
 use surface_fitting_int
 use rand_int
 use norm_int
 implicit none
 integer :: i, j, q, kb
 integer, parameter :: ngrid = 9, nord = 4, ndegree = nord - 1, ndata = 25,          &
       nbkpt = ngrid + 2 * ndegree, nvalues = 50, kv = 2, nvalkv = nvalues / kv
 real(kind(1d0)), parameter :: zero = 0d0, one = 1d0
 real(kind(1d0)), parameter :: tolerance = 1d-3
 real(kind(1d0)), target :: spline_data (4, ndata), bkptx(nbkpt),                    &
       bkpty(nbkpt), coeff(ngrid + ndegree - 1, ngrid + ndegree - 1),                &
       x(nvalues), y(nvalues), values(nvalues, nvalues), delta
 real(kind(1d0)), pointer :: pointer_bkpt(:)
 type(d_spline_knots) knotsx, knotsy
 type(d_surface_constraints), allocatable :: c(:)
 ! Данные, задающие поверхность
 real(kind(1e0)) :: data (3 * ndata) =                                             &
       (/ 2.0, 15.0, 2.5, 2.49, 7.647, 3.2, 2.981, 0.291, 3.4, 3.471, -7.062, 3.5,  &
       3.961, -14.418, 3.5, 7.45, 12.003, 2.5, 7.35, 6.012, 3.5, 7.251, 0.018, 3.0, &
       7.151, -5.973, 2.0, 7.051, -11.967, 2.5, 10.901, 9.015, 2.0, 10.751,         &
       4.536, 1.925, 10.602, 0.06, 1.85, 10.453, -4.419, 1.576, 10.304, -8.895, 1.7, &
       14.055, 10.509, 1.5, 14.194, 6.783, 1.3, 14.331, 3.054, 1.7, 14.469, -0.672, &
       2.1, 14.607, -4.398, 1.75, 15.0, 12.0, 0.5, 15.729, 8.067, 0.5, 16.457, 4.134, &
       0.7, 17.185, 0.198, 1.1, 17.914, -3.735, 1.7 /)
 ! Массив, введенный для графического отображения B-сплайна
 real(4), allocatable :: Bspline(:, :)
 !dec$attributes array_visualizer :: Bspline
 allocate(Bspline(3, nvalkv * nvalkv))
 spline_data(1:3, :) = reshape(data, (/ 3, ndata /)); spline_data(4, :) = one
 ! Задаем узлы поиска B-сплайна, находя
 ! предварительно границы области аппроксимации
 bkptx(1:ndegree) = minval(spline_data(1, :))
 bkptx(nbkpt - ndegree + 1:nbkpt) = maxval(spline_data(1, :))
 delta = (bkptx(nbkpt) - bkptx(ndegree)) / (ngrid - 1)
 bkptx(nord:nbkpt - ndegree) = (/ (bkptx(1) + i * delta, i = 0, ngrid - 1) /)
 ! Задаем степень B-сплайна по оси x и вычисляем knotsx
 pointer_bkpt => bkptx
 knotsx = d_spline_knots(ndegree, pointer_bkpt)
 bkpty(1:ndegree) = minval(spline_data(2, :))
 bkpty(nbkpt - ndegree + 1:nbkpt) = maxval(spline_data(2, :))
 delta = (bkpty(nbkpt) - bkpty(ndegree)) / (ngrid - 1)
 bkpty(nord:nbkpt - ndegree) = (/ (bkpty(1) + i * delta, i = 0, ngrid - 1) /)
```

```
! Задаем степень B-сплайна по оси x и вычисляем knotsy
pointer_bkpt => bkpty
knotsy = d_spline_knots(ndegree, pointer_bkpt)
! Выполняем сглаживание и получаем коэффициенты B-сплайна
coeff = surface_fitting(spline_data, knotsx, knotsy)
delta = (bkptx(nbkpt) - bkptx(1)) / (nvalues + 1)
x = (/ (bkptx(1) + i * delta, i = 1, nvalues) /)
delta = (bkpty(nbkpt) - bkpty(1)) / (nvalues + 1)
y = (/ (bkpty(1) + i * delta, i = 1, nvalues) /)
! Оцениваем функцию на прямоугольной сетке
values = surface_values((/ 0, 0 /), x, y, knotsx, knotsy, coeff)
! Графическое отображение результата - сплайна без ограничений -
! приведено на рис. 2.8, a
kb = 1
do i = 1, nvalues, kv
  Bspline(1, kb:kb + nvalkv - 1) = x(i)
  Bspline(2, kb:kb + nvalkv - 1) = y(1:nvalues:kv)
  Bspline(3, kb:kb + nvalkv - 1) = values(i, 1:nvalues:kv)
  kb = kb + nvalkv
end do                                  ! Текст подпрограммы vGraph2 см. выше
! Вызов ОМ
call vGraph2(Bspline, nvalkv * nvalkv)
! Вычисляем число элементов массива values, меньших или равных нулю
! Затем налагаем ограничения на сплайн таким образом, чтобы его z-координата
! была не меньше допуска в тех точках, где она меньше или равна нулю
q = count(values <= zero)
allocate(c(q))
do i = 1, nvalues
 do j = 1, nvalues
  if(values(i, j) <= zero) then
    c(q) = surface_constraints(point = (/ x(i), y(j) /), type = '>=', value = tolerance)
    q = q - 1
  end if
 end do
end do
! Выполняем сглаживание и получаем коэффициенты B-сплайна
coeff = surface_fitting(spline_data, knotsx, knotsy, constraints = c)
deallocate(c)
! Оцениваем поверхность в узлах сетки и проверяем, есть ли неположительные
! значения в массиве values (на самом деле такие должны отсутствовать)
values = surface_values((/ 0, 0 /), x, y, knotsx, knotsy, coeff)
if(count(values <= zero) == 0) then
  write(*, *) 'Example 4 for SURFACE_FITTING is correct.'
```

```
 end if
 ! Графическое отображение результата - сплайна с ограничениями -
 ! приведено на рис. 2.8, б
 deallocate(Bspline); allocate(Bspline(3, nvalkv * nvalkv))
 kb = 1
 do i = 1, nvalues, kv
   Bspline(1, kb:kb + nvalkv - 1) = x(i)
   Bspline(2, kb:kb + nvalkv - 1) = y(1:nvalues:kv)
   Bspline(3, kb:kb + nvalkv - 1) = values(i, 1:nvalues:kv)
   kb = kb + nvalkv
 end do                                  ! Текст подпрограммы vGraph2 см. выше
 ! Вызов ОМ
 call vGraph2(Bspline, nvalkv * nvalkv)
 deallocate(Bspline)
end program ex_suf4
```



а *б*

Рис. 2.8. В-сплайн: а - часть поверхности лежит ниже плоскости z = 0; б - после задания ограничений сплайн не имеет неположительных z-координат

3. ИНТЕГРИРОВАНИЕ И ДИФФЕРЕНЦИРОВАНИЕ

3.1. ВВЕДЕНИЕ

3.1.1. КВАДРАТУРЫ С ОДНОЙ ПЕРЕМЕННОЙ

Первые 9 из приводимых в настоящей главе процедур библиотеки IMSL приближенно вычисляют интеграл

$$\int_a^b f(x)w(x)dx.$$

Весовая функция $w(x)$ используется для учета известных особенностей (алгебраических или логарифмических), осцилляций или для указания того, что требуется вычислить интеграл Коши. В общем случае рекомендуется употреблять подпрограмму QDAGS (даже при отсутствии особенностей в конечных точках). Если нужна большая эффективность, то следует применить подпрограмму QDAG или QDAG*.

Весовая функция

- $w(x) = 1$ в подпрограммах QDAGS, QDAG, QDAGP, QDAGI и QDNG;
- $w(x) = \sin\omega x$ или $w(x) = \cos\omega x$ в подпрограммах QDAWO (для конечного интервала) и QDAWF (для бесконечного интервала);
- $w(x) = (x - a)^{\alpha}(b - x)^{\beta}\ln(x - a)\ln(b - x)$ в подпрограмме QDAWS (сомножитель ln является необязательным);
- $w(x) = 1/(x - c)$ в подпрограмме QDAWC, возвращающей значение интеграла Коши.

Интерфейсы этих процедур весьма схожи: интегрируемая функция имеет имя *f*; нижняя и верхняя границы интегрирования - соответственно имена *a* и *b*. Задаваемые абсолютная ошибка ε и относительная ошибка ρ - соответственно имена *errabs* и *errrel*. Возвращают эти процедуры две величины: *result* и *errest*, означающие соответственно примерное значение интеграла *r* и оценку ошибки интегрирования *e*. При этом справедливо неравенство

$$\left|\int_a^b f(x)w(x)dx - r\right| \le e \le \max\left(\varepsilon, \rho\left|\int_a^b f(x)w(x)dx\right|\right).$$

Если данные заданы в табличной форме, то для интегрирования предварительно находится интерполирующая кривая, а затем выполняется ее интегрирование. Причем такой подход может быть осуществлен, например, в результате употребления процедуры CSINT, вычисляющей интерполяционный кубический сплайн с граничными условиями "нет узла", или BSINT, находящей интерполяционный B-сплайн заданного порядка, а затем функции PPITG, возвращающей интеграл одномерного сплайна, используя его кусочно-многочленное представление.

3.1.2. КВАДРАТУРЫ С НЕСКОЛЬКИМИ ПЕРЕМЕННЫМИ

Подпрограмма TWODQ возвращает приблизительное значение двойного интеграла

$$\int_a^b \int_{g(x)}^{h(x)} f(x,y)dydx.$$

Интеграл функции n переменных

$$\int_{a_1}^{b_1} \cdots \int_{a_n}^{b_n} f(x_1,\ldots,x_n)dx_n \ldots dx_1$$

вычисляется подпрограммой QAND.

Если данные заданы в виде двумерной или трехмерной таблицы, то для вычисления по этим данным интеграла предварительно подпрограммой BS2IN или BS3IN находится интерполяционный B-сплайн, а затем функцией BS2IG или BS3IG выполняется его интегрирование.

3.1.3. ПРАВИЛА ГАУССА И ТРЕХЭЛЕМЕНТНЫЕ РЕКУРРЕНТНЫЕ СООТНОШЕНИЯ

Процедура GQRUL, используя квадратурные правила Гаусса, при выбранной из табл. 3.1 весовой функции вырабатывает точки x_i и веса w_i для $i = 1, \ldots, n$, такие, что

$$\int_a^b f(x)w(x)dx \approx \sum_{i=1}^{n} f(x_i)w_i$$

для всех функций f, являющихся многочленами, степень которых меньше чем $2n$.

Таблица 3.1. Квадратурные весовые функции

w(x)	*Интервал*	*Весовая функция*
1	(-1, 1)	Лежандра
$1/\sqrt{1-x^2}$	(-1, 1)	Чебышева 1-го рода
$\sqrt{1-x^2}$	(-1, 1)	Чебышева 2-го рода
e^{-x^2}	(-∞, ∞)	Эрмита
$(1+x)^{\alpha}(1-x)^{\beta}$	(-1, 1)	Якоби
$e^{-x}x^{\alpha}$	(0, ∞)	Обобщенная Лагерра
$1/\cosh(x)$	(-∞, ∞)	Гиперболический косинус

Где возможно, подпрограмма GQRUL применяет Гаусса - Радау или Гаусса - Лобатто квадратурные правила.

Подпрограмма RECCF формирует трехэлементные рекуррентные соотношения для ортогональных многочленов с учетом весовой функции, выбираемой из табл. 3.1.

Подпрограмма GQRCF вырабатывает квадратурное правило Гаусса, Гаусса - Радау или Гаусса - Лобатто по известному трехэлементному рекуррентному соотношению для ортогональных многочленов. Подпрограмма RECQR является обратной к подпрограмме GQRCF, поскольку вычисляет коэффициенты рекуррентного соотношения по заданной квадратурной формуле Гаусса.

Последняя подпрограмма раздела (FQRUL) генерирует квадратурные правила Фейера с весовыми функциями:

- $w(x) = 1$;
- $w(x) = 1/(x - \alpha)$;
- $w(x) = (b - x)^{\alpha}(x - a)^{\beta}$;
- $w(x) = (b - x)^{\alpha}(x - a)^{\beta}\ln(x - a)$;
- $w(x) = (b - x)^{\alpha}(x - a)^{\beta}\ln(b - x)$.

3.1.4. ЧИСЛЕННОЕ ДИФФЕРЕНЦИРОВАНИЕ

Выполняется функцией DERIV. Функция выполняет оценку первой, второй или третьей производной предоставленной пользователем функции.

3.2. ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ ФУНКЦИИ ОДНОЙ ПЕРЕМЕННОЙ

3.2.1. ПЕРЕЧЕНЬ И ПАРАМЕТРЫ ПРОЦЕДУР

Перечень процедур дан в табл. 3.2, а их параметры - в табл. 3.3. Дополнительные параметры процедур второго уровня приводятся в табл. 3.4.

Таблица 3.2. Подпрограммы, выполняющие двумерные и трехмерные комплексные быстрые преобразования Фурье

Подпрограмма	*Назначение*
QDAGS	Вычисляет интеграл функции, которая может иметь особенности в конечных точках
QDAG	Вычисляет интеграл функции, используя глобальную адаптивную схему Гаусса - Кронрода
QDAGP	Вычисляет интеграл функции с особенностями в заданных точках
QDAGI	Вычисляет интеграл функции на бесконечном или полубесконечном интервале
QDAWO	Вычисляет интеграл функции, содержащей синус или косинус
QDAWF	Вычисляет интеграл Фурье
QDAWS	Вычисляет интеграл функции с алгебраическо-логарифмическими особенными точками
QDAWC	Вычисляет интеграл функции $f(x)/(x - c)$
QDNG	Вычисляет интеграл гладкой функции, не прибегая к адаптивной схеме

Таблица 3.3. Параметры подпрограмм из табл. 3.2

Имя	*Смысл/вид*	*Тип*
a	Нижняя граница интегрирования / *входной*	REAL(4) или REAL(8)
b	Верхняя граница интегрирования; $b \geq a$ / *входной*	То же
errabs	Желаемая абсолютная ошибка / *входной*	"
errrel	Желаемая относительная ошибка / *входной*	"
errest	Оценка величины абсолютной ошибки численного интегрирования / *выходной*	"
f	Пользовательская функция, интеграл которой должен быть вычислен. Имеет вид $f(x)$, где x - независимая переменная. Должна в вызывающей программе получить атрибут EXTERNAL / *пользовательская функция*	"

iweigh	Тип весовой функции $w(x)$; если • *iweigh* = 1, то $w(x) = \cos(omega * x)$; • *iweigh* = 2, то $w(x) = \sin(omega * x)$ / *входной*	INTEGER(4)
omega	Параметр весовой функции / *входной*	REAL(4) или REAL(8)
result	Приблизительное значение интеграла функции на отрезке [*a, b*] / *выходной*	То же

Таблица 3.4. Дополнительные параметры подпрограмм второго уровня

Имя	*Смысл/вид*	*Тип*
alist	Вектор размера *maxsub*, содержащий список из *nsubin* левых точек подынтервалов / *выходной*	REAL(4) или REAL(8)
blist	Вектор размера *maxsub*, содержащий список из *nsubin* правых точек подынтервалов / *выходные*	То же
elist	Вектор размера *maxsub*, содержащий оценки ошибок *nsubin* значений вектора *rlist* / *выходной*	"
iord	Вектор размера *maxsub*. Первые *k* элементов вектора содержат номера оценок ошибок на подынтервалах, такие, что последовательность *elist*(*iord*(1)), …, *elist*(*iord*(*k*)) является убывающей, где *k* = *nsubin*, если *nsubin* ≤ (*maxsub*/2 + 2), и *k* = *maxsub* + 1 - *nsubin* - в противном случае / *выходной*	INTEGER(4)
maxcby	Верхняя граница для номера многочлена Чебышева; подпрограммы QDAWO и QDAWF используют *maxcby*=21	"
maxsub	Число разрешенных подынтервалов; подпрограммы QDAGS, QDAG, QDAWI, QDAWS и QDAWC используют 500 подынтервалов; QDAGP - 450; QDAWO - 390; QDAWF - 365; TWODQ - 250 / *входной*	"
neval	Число оценок интегрируемой функции *f* / *выходной*	"
nsubin	Число сгенерированных подынтервалов / *выходной*	"
rlist	Вектор размера *maxsub*, содержащий приближенные значения *nsubin* интегралов на интервалах, определяемых векторами *alist* и *blist* / *выходной*	REAL(4) или REAL(8)

3.2.2. ПОДПРОГРАММА QDAGS (DQDAGS)

Вычисляет интеграл функции, которая может иметь особенности в конечных точках. Имеет вызов

CALL QDAGS(*f, a, b, errabs, errrel, result, errest*)

Описание параметров см. в табл. 3.3.

Автоматически для решения предоставляется память:

- 2500 байт в случае QDAGS;
- 4500 байт в случае DQDAGS.

Память можно выделить явно, употребив Q2AGS (DQ2AGS):

CALL Q2AGS(*f, a, b, errabs, errrel, result, errest,* &
maxsub, neval, nsubin, alist, blist, rlist, elist, iord)

В табл. 3.4 см. смысл дополнительных параметров подпрограммы Q2AGS.

Комментарии:

1. При работе с QDAGS могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
4	1	Достигнуто максимально допустимое число подынтервалов
3	2	Ошибка округления не позволяет обеспечить заданную точность
3	3	Обнаружена потеря точности
3	4	Ошибка округления в экстраполяционной таблице не позволяет обеспечить заданную точность
4	5	Интеграл, возможно, расходится или медленно сходится

2. Пусть *exact* является точным значением интеграла; подпрограмма пытается найти результат *result* такой, что $|exact - result| \le \max(errabs, errrel * |exact|)$. Чтобы задать только относительную ошибку, положите *errabs* = 0. Аналогично для задания одной абсолютной ошибки установите *errrel* = 0.

Описание:

Подпрограмма QDAGS использует адаптивную схему, уменьшающую абсолютную ошибку. Подпрограмма делит отрезок [*a*, *b*] на подынтервалы и использует 21-точечное правило Гаусса - Кронрода для оценки интеграла на каждом подынтервале. Оценка ошибки на каждом подынтервале выполняется путем сравнения результата с величиной, получаемой 10-точечным квадратурным правилом Гаусса.

Подпрограмма может вычислять интегралы функций, имеющих особенности в конечных точках. Разумеется, она дает хорошие результаты и для функций, не имеющих таких особенностей. В дополнение к общей стратегии, изложенной ниже при рассмотрении подпрограммы QDAG, подпрограмма QAGS использует экстраполяционную процедуру, известную как ε-алгоритм.

Подпрограмма QDAGS является реализацией процедуры QAGS; полностью документированной в [45]. Если QDAGS не удается получить приемлемый результат, то следует испытать подпрограмму QDAG или QDAG*.

Пример. Оценивается интеграл

$$\int_0^1 \ln(x) x^{-1/2} dx = -4.$$

```
program qdagsTest
 use dfimsl
 real(4) :: a, b, errabs, errest, error, errrel, exact, f, result
 external f
 a = 0.0; b = 1.0                          ! Границы интегрирования
 errabs = 0.0; errrel = 0.001              ! Допускаемые ошибки
 call qdags(f, a, b, errabs, errrel, result, errest)
 exact = -4.0; error = abs(result - exact)
 write(*, 99999) result, exact, errest, error
 99999 format(' Computed =', f8.3, 13x, ' Exact =', f8.3, /, /,                    &
       ' Error estimate =', 1pe10.3, 6x, ' Error =', 1pe10.3)
end program qdagsTest

real function f(x)
 real(4) :: x
 f = alog(x) / sqrt(x)
end function f
```

Результат:

```
Computed = -4.000                 Exact = -4.000
Error estimate = 2.599E-04        Error = 4.792E-06
```

3.2.3. ПОДПРОГРАММА QDAG (DQDAG)

Вычисляет интеграл функции, используя глобальную адаптивную схему Гаусса - Кронрода. Имеет вызов

CALL QDAG(*f, a, b, errabs, errrel, irule, result, errest*)

Описание параметров, кроме выходного параметра *irule*, см. в табл. 3.3.

irule - квадратурное правило Гаусса - Кронрода использует:

- 7-15 точек, если *irule* = 1;
- 10-21 точку, если *irule* = 2;
- 15-31 точку, если *irule* = 3;
- 20-41 точку, если *irule* = 4;

- 25-51 точку, если *irule* = 5;
- 30-61 точку, если *irule* = 6.

Рекомендуемое для большинства функций значение *irule* = 2. Если функция имеет пиковую особенность, используйте *irule* = 1. В случае осциллирующей функции задавайте *irule* = 6.

Автоматически для решения предоставляется память:

- 2500 байт в случае QDAG;
- 4500 байт в случае DQDAG.

Память можно выделить явно, употребив Q2AG (DQ2AG):

```
CALL Q2AG(f, a, b, errabs, errrel, irule, result, errest,          &
     maxsub, neval, nsubin, alist, blist, rlist, elist, iord)
```

Смысл дополнительных параметров подпрограммы Q2AG см. в табл. 3.4.

Комментарии:

1. При работе с QDAGS могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	1	Достигнуто максимально допустимое число подынтервалов
3	2	Ошибка округления не позволяет обеспечить заданную точность
3	3	Обнаружена потеря точности

2. См. комментарий 2 в предшествующем разделе.

Описание:

Подпрограмма QDAG является интегратором общего назначения, использующим с целью уменьшения ошибки глобальную адаптивную схему. Подпрограмма делит отрезок [*a*, *b*] на подынтервалы и использует ($2k$ + 1)-точечное Гаусса - Кронрода правило для оценки интеграла на каждом из подынтервалов. Оценка ошибки на каждом подынтервале выполняется путем сравнения результата с величиной, получаемой *k*-точечным квадратурным правилом Гаусса. Подынтервал с наибольшей ошибкой делится затем пополам, и на каждой части применяется та же процедура. Процесс деления продолжается, пока не удовлетворен допуск на ошибку, или не обнаружена ошибка округления, или подынтервал оказался чрезмерно маленьким, или достигнуто максимальное число подынтервалов.

Подпрограмма QDAG основана на процедуре QAG, приведенной в [45]. Если QDAGS не удается получить приемлемый результат, то следует испытать подпрограмму QDAG*.

Пример. Оценивается интеграл

$$\int_0^2 xe^x dx = e^2 + 1.$$

Поскольку интеграл не является осциллирующим, используется *irule* = 1.

```
program qdagTest
 use dfimsl
 integer(4) :: irule
 real(4) :: a, b, errabs, errest, error, errrel, exact, f, result
 external f
 a = 0.0; b = 2.0                    ! Границы интегрирования
 errabs = 0.0; errrel = 0.001        ! Допускаемые ошибки
 irule = 1                           ! Правило для неосциллирующей функции
 call qdag(f, a, b, errabs, errrel, irule, result, errest)
 exact = 1.0 + exp(2.0); error = abs(result - exact); error = abs(result - exact)
 write(*, 99999) result, exact, errest, error
 99999 format(' Computed =', f8.3, 13x, ' Exact =', f8.3, /, /,                &
       ' Error estimate =', 1pe10.3, 6x, ' Error =', 1pe10.3)
end program qdagTest

real function f(x)
 real(4) :: x
 f = x * exp(x)
end function f
```

Результат:

```
Computed = 8.389                Exact = 8.389
Error estimate = 5.000E-05      Error = 0.000E+00
```

3.2.4. ПОДПРОГРАММА QDAGP (DQDAGP)

Вычисляет интеграл функции с особенностями в заданных точках. Имеет вызов

CALL QDAGP(*f, a, b, npts, points, errabs, errrel, result, errest*)

Описание параметров, кроме *входных* параметров *npts* и *points*, см. в табл. 3.3.

npts - число точек раздела.

points - вектор размера *npts*, содержащий точки раздела на отрезке интегрирования. Обычно в этих точках интегрируемая функция имеет особенности.

Автоматически для решения предоставляется память:

- 2704 + 2*npts* байт в случае QDAGP;
- 4506 + 3*npts* байт в случае DQDAGP.

Память можно выделить явно, употребив Q2AGP (DQ2AGP):

CALL Q2AGP(*f, a, b, npts, points, errabs, errrel, result, errest,* &
maxsub, neval, nsubin, alist, blist, rlist, elist, iord, Level, wk, iwk)

Смысл дополнительных параметров подпрограммы Q2AGP, кроме *входного* параметра *Level* и *рабочих векторов wk* и *iwk*, см. в табл. 3.4.

Level - вектор размера *maxsub*, содержащий уровни подынтервалов. Подынтервал (*aa*, *bb*), лежащий в пределах (*p*1, *p*2), где *p*1 и *p*2 - предоставленные пользователем точки раздела или пределы интегрирования, имеет уровень *L*, если ABS(*bb* - *aa*) = 2^{-L}ABS(*p*2 - *p*1).

wk, iwk - рабочие векторы размера *npts* + 2.

Комментарии:

1. При работе с QDAGP могут возникать те же, что и при работе с подпрограммой QDAGS, информационные ошибки.
2. См. комментарий 2 в разд. 3.2.2.

Описание:

Подпрограмма QDAGP использует глобальную адаптивную схему, позволяющую уменьшать абсолютную ошибку. Первоначально отрезок [*a*, *b*] подразделяется на *npts* + 1 предоставленных пользователем подынтервала и применяется 21-точечное Гаусса - Кронрода правило для оценки интеграла на каждом подынтервале. Оценка ошибки на каждом подынтервале выполняется путем сравнения результата с величиной, получаемой 10-точечным квадратурным правилом Гаусса.

Заметьте, что QDAGP никогда не оценивает интегрируемую функцию в предоставленных пользователем точках раздела.

Подпрограмма QDAGP рассматривает точку раздела как внутреннюю особенность. В дополнение к общей стратегии, рассматриваемой при описании подпрограммы QDAG, подпрограмма QDAGP использует экстраполяционную процедуру, известную как ε-алгоритм. Подпрограмма QDAGP основана на приведенной в [45] процедуре QAGP.

Пример. Задаются две точки раздела - 1.0 и 2.0, в которых функция имеет особенности, и оценивается интеграл

$$\int_0^3 x^3 \ln\left|(x^2-1)(x^2-2)\right| dx = 61\ln 2 + \frac{77}{4}\ln 7 - 27 .$$

```
program qdagpTest
 use dfimsl
 integer(4) :: npts
 real(4) :: a, b, errabs, errest, error, errrel, exact, f, result, points(2)
 external f
 a = 0.0; b = 3.0                    ! Границы интегрирования
 errabs = 0.0; errrel = 0.01         ! Допускаемые ошибки
 npts = 2                            ! Особенности
 points(1) = 1.0; points(2) = sqrt(2.0)
 call qdagp(f, a, b, npts, points, errabs, errrel, result, errest)
 exact = 61.0 * alog(2.0) + 77.0 / 4.0 * alog(7.0) - 27.0
 error = abs(result - exact)
 write(*, 99999) result, exact, errest, error
 99999 format(' Computed =', f8.3, 13x, ' Exact =', f8.3, /, /,                &
       ' Error estimate =', 1pe10.3, 6x, ' Error =', 1pe10.3)
end program qdagpTest

real function f(x)
 real(4) :: x
 f = x**3 * alog(abs((x * x - 1.0) * (x * x - 2.0)))
end function f
```

Результат:

```
Computed = 52.741                Exact = 52.741
Error estimate = 5.062E-01       Error = 6.218E-04
```

3.2.5. ПОДПРОГРАММА QDAGI (DQDAGI)

Вычисляет интеграл функции на бесконечном или полубесконечном интервале. Имеет вызов

CALL QDAGI(*f, bound, interv, errabs, errrel, result, errest*)

Описание параметров, кроме *входных* параметров *bound* и *interv*, см. в табл. 3.3.

bound - конечная граница диапазона интегрирования. Игнорируется, если *interv* = 2.

interv - флаг, указывающий на интервал интегрирования. Интегрирование выполняется на интервале:

- (-∞, *bound*), если *interv* = -1;
- (*bound*, +∞), если *interv* = 1;
- (-∞, +∞), если *interv* = 2.

Автоматически для решения предоставляется память:

- 2500 байт в случае QDAGI;
- 4500 байт в случае DQDAGI.

Память можно выделить явно, употребив Q2AGI (DQ2AGI):

CALL Q2AGI(*f, bound, interv, errabs, errrel, result, errest,* &
maxsub, neval, nsubin, alist, blist, rlist, elist, iord)

Смысл дополнительных параметров подпрограммы Q2AGI см. в табл. 3.4.

Комментарии:

1. При работе с QDAGI могут возникать те же, что и при работе с подпрограммой QDAGS, информационные ошибки.
2. См. комментарий 2 в разд. 3.2.2.

Описание:

Подпрограмма QDAGI использует глобальную адаптивную схему, позволяющую уменьшать абсолютную ошибку. Первоначально бесконечный или полубесконечный интервал преобразовывается в отрезок [0, 1]. Затем QDAGI, употребляя 21-точечное Гаусса - Кронрода правило, оценивает интеграл и ошибку. Каждый подынтервал с неприемлемой ошибкой подвергается делению пополам, которое продолжается, пока не удовлетворен допуск на ошибку или не возникли завершающие условия (обнаружена ошибка округления, или подынтервал оказался чрезмерно маленьким, или достигнуто максимальное число подынтервалов). Подпрограмма может вычислять интегралы функций, имеющих особенности в конечных точках. В дополнение к общей стратегии, изложенной ниже при рассмотрении подпрограммы QDAG, подпрограмма QDAGI использует экстраполяционную процедуру, известную как ε-алгоритм.

Подпрограмма QDAGI является реализацией процедуры QAGI, полностью документированной в [45].

Пример. Вычисляется интеграл

$$\int_0^{\infty} \frac{\ln x}{1+100x^2}\,dx = \frac{-\pi \ln(10)}{20}.$$

```
program qdagiTest
 use dfimsl
 integer(4) :: interv
 real(4) :: a, b, bound, errabs, errest, error, errrel, exact, f, result
```

```
 external f
 bound = 0.0; interv = 1                ! Границы интегрирования
 errabs = 0.0; errrel = 0.001           ! Допускаемые ошибки
 call qdagi(f, bound, interv, errabs, errrel, result, errest)
 exact = -const('pi') * alog(10.0) / 20.0
 error = abs(result - exact)
 write(*, 99999) result, exact, errest, error
 99999 format(' Computed =', f8.3, 13x, ' Exact =', f8.3, /, /,                &
       ' Error estimate =', 1pe10.3, 6x, ' Error =', 1pe10.3)
end program qdagiTest

real function f(x)
 real(4) :: x
 f = alog(x) / (1.0 + (10.0 * x)**2)
end function f
```

Результат:

Computed = -0.362 Exact = -0.362
Error estimate = 3.248E-06 Error = 2.980E-08

3.2.6. ПОДПРОГРАММА QDAWO (DQDAWO)

Вычисляет интеграл функции, содержащей синус или косинус. Имеет вызов

CALL QDAWO(*f, a, b, iweigh, omega, errabs, errrel, result, errest*)

Описание параметров, кроме *выходного* параметра *result*, см. в табл. 3.3.

result - оценка интеграла на отрезке [*a, b*] функции *f* * *w*(*x*).

Автоматически для решения предоставляется память:

- 2865 байт в случае QDAWO;
- 4950 байт в случае DQDAWO.

Память можно выделить явно, употребив Q2AWO (DQ2AWO):

CALL Q2AWO(*f, a, b, iweigh, omega, errabs, errrel, result, errest,* &
maxsub, maxcby, neval, nsubin, alist, blist, rlist, elist, iord, nnlog, wk)

Смысл дополнительных параметров подпрограммы Q2AWO, кроме *выходного* вектора *nnlog* и *рабочего* вектора *wk*, см. в табл. 3.4.

nnlog - вектор размера *maxsub*, содержащий уровни подынтервалов. Равенство *nnlog*(*i*) = *L* означает, что подынтервал с номером *i* имеет длину $ABS(b - a)^{1-L}$.

wk - рабочий вектор размера 25*maxcby*.

Комментарии:

1. При работе с QDAWO могут возникать те же, что и при работе с подпрограммой QDAGS, информационные ошибки.
2. См. комментарий 2 в разделе с описанием подпрограммы QDAGS.

Описание:

Подпрограмма QDAGP использует глобальную адаптивную схему, позволяющую уменьшать абсолютную ошибку. Подпрограмма вычисляет интеграл специального вида $w(x)f(x)$, где $w(x) = \cos\omega x$ или $w(x) = \sin\omega x$. В зависимости от соотношения длины подынтервала и частоты ω подпрограмма применяет для аппроксимации интеграла на подынтервале либо модифицированную процедуру Кленшо - Куртиса, либо 7-15-точечное Гаусса - Кронрода правило. В дополнение к общей стратегии, рассматриваемой при описании подпрограммы QDAG, подпрограмма QDAWO использует экстраполяционную процедуру, известную как ε-алгоритм. Подпрограмма QDAWO основана на приведенной в [45] процедуре QAWO.

Пример. Вычисляется интеграл

$$\int_0^1 \ln(x)\sin(10\pi x)dx.$$

Заметьте, что логарифмическая функция кодируется таким образом, что устраняется особенность в точке $x = 0$.

```
program qdawoTest
 use dfimsl
 integer(4) :: iweigh
 real(4) :: a, b, errabs, errest, error, errrel, exact, f, omega, result
 external f
 a = 0.0; b = 1.0                        ! Границы интегрирования
 iweigh = 2; omega = 10.0* const('pi')   ! Весовая функция sin(10πx)
 errabs = 0.0; errrel = 0.001            ! Допускаемые ошибки
 call qdawo(f, a, b, iweigh, omega, errabs, errrel, result, errest)
 exact = -0.1281316; error = abs(result - exact)
 write(*, 99999) result, exact, errest, error
 99999 format(' Computed =', f8.3, 13x, ' Exact =', f8.3, /, /,                &
      ' Error estimate =', 1pe10.3, 6x, ' Error =', 1pe10.3)
end program qdawoTest

real function f(x)
 real(4) :: x
 if(x == 0.) then; f = 0.0; else; f = alog(x); end if
end function f
```

Результат:

Computed = -0.128 Exact = -0.128
Error estimate = 7.504E-05 Error = 5.275E-06

3.2.7. ПОДПРОГРАММА QDAWF (DQDAWF)

Вычисляет интеграл Фурье. Имеет вызов

CALL QDAWF(*f, a, iweigh, omega, errabs, result, errest*)

Описание параметров см. в табл. 3.3.

Автоматически для решения предоставляется память:

- 2865 байт в случае QDAWF;
- 4950 байт в случае DQDAWF.

Память можно выделить явно, употребив Q2AWF (DQ2AWF):

```
CALL Q2AWF(f, a, iweigh, omega, errabs, result, errest,          &
     maxcyl, maxsub, maxcby, neval, ncycle, rslist, erlist,      &
     ierlst, nsubin, wk, iwk)
```

Смысл дополнительных параметров подпрограммы Q2AWF, кроме приведенных ниже, см. в табл. 3.4.

maxcyl - максимально допустимое число циклов. Является *входным*; не может быть менее трех. Подпрограмма QDAWF использует 50.

ncycle - число выполненных циклов. Является *выходным*.

rslist - вектор размера *maxcyl*, содержащий вклады в интеграл на интервалах ($a + (k - 1) * c$, $a + k * c$), для $k = 1, \ldots,$ *ncycle*. Является *выходным;* $c = (2 * \mathrm{INT}(\mathrm{ABS}(omega)) + 1) * \pi/\mathrm{ABS}(omega)$.

erlist - вектор размера *maxcyl*, содержащий оценки ошибок на интервалах, определенных при описании параметра *rslist*. Является *выходным*.

ierlst - вектор размера *maxcyl*, содержащий флаги ошибок на интервалах, определенных при описании параметра *rslist*. Является *выходным*. Элементы вектора принимают значения:

- 1 - максимальное число разбиений *maxsub* достигнуто на *k*-м цикле;
- 2 - ошибка округления, не позволяет обеспечить заданную точность на *k*-м цикле;
- 3 - чрезвычайно плохое поведение интегрируемой функции в некоторой точке на *k*-м цикле;

- 4 - процедура интегрирования не сходится к заданной точности из-за ошибки округления в экстраполяционной процедуре на *k*-м цикле. Полученный результат выдается в качестве лучшего;
- 5 - интеграл на *k*-м цикле расходится или медленно сходится.

wk - вещественный рабочий вектор размера $4maxsub + 25maxcby$.

iwk - целочисленный рабочий вектор размера $2maxsub$.

Комментарии:

1. При работе с QDAGS могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	1	Плохое поведение интегрируемой функции наблюдается в одном или более циклах
4	2	Достигнуто максимально допустимое число циклов
3	3	Экстраполяционная таблица, созданная для ускорения сходимости серий, формирующих интеграл, не сходится к требуемой точности

2. Пусть *exact* является точным значением интеграла; подпрограмма QDAGS пытается найти результат *result* такой, что $|exact - result| \le errabs$.

Описание:

Подпрограмма QDAWF использует глобальную адаптивную схему, позволяющую уменьшать абсолютную ошибку. Подпрограмма вычисляет интегралы функций, имеющих вид $w(x)f(x)$, где $w(x) = \cos\omega x$ или $w(x) = \sin\omega x$. Интервал интегрирования всегда полубесконечный, имеющий вид $[a, \infty]$. Такие интегралы Фурье аппроксимируются повторными вызовами рассмотренной выше подпрограммы QDAWO и последующей экстраполяцией. Подпрограмма QDAWF является реализацией процедуры QAWF, полностью документированной в [45].

Пример. Вычисляется интеграл

$$\int_0^{\infty} x^{-1/2}\cos(\pi x/2)dx = 1.$$

Функция f кодируется таким образом, что устраняется особенность в точке $x = 0$.

```
program qdawfTest
 use dfimsl
 integer(4) :: iweigh
 real(4) :: a, errabs, errest, error, exact, f, omega, result
```

```
external f
a = 0.0                                  ! Нижняя граница интегрирования
iweigh = 1; omega = 0.5 * const('pi')    ! Весовая функция cos(πx/2)
errabs = 0.001                           ! Требуемая точность результата
call qdawf(f, a, iweigh, omega, errabs, result, errest)
exact = 1.0; error = abs(result - exact)
write(*, 99999) result, exact, errest, error
99999 format(' Computed =', f8.3, 13x, ' Exact =', f8.3, /, /,                    &
      ' Error estimate =', 1pe10.3, 6x, ' Error =', 1pe10.3)
end program qdawfTest

real function f(x)
real(4) :: x
if(x > 0.0) then
  f = 1.0 / sqrt(x)
else
  f = 0.0
end if
end function f
```

Результат:

Computed = 1.000 Exact = 1.000
Error estimate = 6.103E-04 Error = 2.921E-06

3.2.8. ПОДПРОГРАММА QDAWS (DQDAWS)

Вычисляет интеграл функции с алгебраическо-логарифмическими особенными точками. Имеет вызов

CALL QDAWS(*f, a, b, iweigh, alpha, beta, errabs, errrel, result, errest*)

Описание параметров, кроме *входных* параметров *iweigh, alpha* и *beta* и выходного параметра *result*, см. в табл. 3.3.

iweigh - тип весовой функции $w(x)$; если:

iweigh = 1, то $w(x) = (x - a)^{alpha} * (b - x)^{beta}$;

iweigh = 2, то $w(x) = (x - a)^{alpha} * (b - x)^{beta} * \ln(x - a)$;

iweigh = 3, то $w(x) = (x - a)^{alpha} * (b - x)^{beta} * \ln(b - x)$;

iweigh = 4, то $w(x) = (x - a)^{alpha} * (b - x)^{beta} * \ln(x - a) * \ln(b - x)$.

alpha - параметр весовой функции; должен быть больше, чем -1.0.

beta - параметр весовой функции; должен быть больше, чем -1.0.

result - оценка интеграла функции $w(x)\,f(x)$ на отрезке $[a, b]$.

Автоматически для решения предоставляется память:

- 2500 байт в случае QDAWS;
- 4500 байт в случае DQDAWS.

Память можно выделить явно, употребив Q2AWS (DQ2AWS):

CALL Q2AWS(*f, a, b, iweigh, alpha, beta, errabs, errrel, result, errest,* &
maxsub, neval, nsubin, alist, blist, rlist, elist, iord)

Смысл дополнительных параметров подпрограммы Q2AWS см. в табл. 3.4.

Комментарии:

1. При работе с QDAWS могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	1	Достигнуто максимально допустимое число подынтервалов
3	2	Ошибка округления не позволяет обеспечить заданную точность
3	3	Обнаружена потеря точности

2. См. комментарий 2 в разд. 3.2.2.

Описание:

Подпрограмма QDAWS использует глобальную адаптивную схему, позволяющую уменьшать абсолютную ошибку. Подпрограмма находит интеграл функции, имеющей вид $w(x)\,f(x)$, где $w(x)$ - одна из приведенных выше весовых функций. Применяется комбинация формул Кленшо - Куртиса и Гаусса - Кронрода. В дополнение к общей стратегии, рассмотренной при описании подпрограммы QDAG, подпрограмма QDAWS использует экстраполяционную процедуру, известную как ε-алгоритм. Подпрограмма QDAWS основана на приведенной в [45] процедуре QAWS.

Пример. Вычисляется интеграл

$$\int_0^1 \sqrt{(1+x)(1-x)}\,x\ln(x)\,dx = \frac{3\ln 2 - 4}{9}.$$

```
program qdawsTest
 use dfimsl, nouse => beta
 integer(4) :: iweigh
 real(4) :: a, alpha, b, beta, errabs, errest, error, errrel, exact, f, result
 external f
 a = 0.0; b = 1.0                    ! Границы интегрирования
 alpha = 1.0; beta = 0.5; iweigh = 2 ! Задаем весовую функцию
 errabs = 0.0; errrel = 0.001        ! Допускаемые ошибки
```

```
 call qdaws(f, a, b, iweigh, alpha, beta, errabs, errrel, result, errest)
 exact = (3.0 * alog(2.0) -4.0) / 9.0
 error = abs(result - exact)
 write(*, 99999) result, exact, errest, error
 99999 format(' Computed =', f8.3, 13x, ' Exact =', f8.3, /, /,                &
       ' Error estimate =', 1pe10.3, 6x, ' Error =', 1pe10.3)
end program qdawsTest

real function f(x)
 real(4) :: x
 f = sqrt(1.0 + x)
end function f
```

Результат:

Computed = -0.213 Exact = -0.213
Error estimate = 0.000E-08 Error = 1.490E-08

3.2.9. ПОДПРОГРАММА QDAWC (DQDAWC)

Вычисляет интеграл функции $f(x)/(x - c)$. Имеет вызов

CALL QDAWC(*f, a, b, c, errabs, errrel, result, errest*)

Описание параметров, кроме *входного* параметра *c* и выходного параметра *result*, см. в табл. 3.3.

c - особенная точка; параметр *c* не должен равняться *a* или *b*.

result - оценка интеграла функции $f(x)/(x - c)$ на отрезке $[a, b]$.

Автоматически для решения предоставляется память:

- 2500 байт в случае QDAWC;
- 4500 байт в случае DQDAWC.

Память можно выделить явно, употребив Q2AWC (DQ2AWC):

CALL Q2AWC(*f, a, b, c, errabs, errrel, result, errest,* &
maxsub, neval, nsubin, alist, blist, rlist, elist, iord)

Смысл дополнительных параметров подпрограммы Q2AWC см. в табл. 3.4.

Комментарии см. в предшествующем разделе.

Описание:

Подпрограмма QDAWC использует глобальную адаптивную схему, пытаясь уменьшить абсолютную ошибку. Подпрограмма находит интеграл функции $w(x)\,f(x)$, где $w(x) = = 1/(x - c)$. Если *c* находится на отрезке интегрирования, то интеграл интерпретируется как главное значение Коши.

Применяется комбинация формул Кленшо - Куртиса и Гаусса - Кронрода. В дополнение к общей стратегии, рассмотренной при описании подпрограммы QDAG, подпрограмма QDAWS использует экстраполяционную процедуру, известную как ε-алгоритм. Подпрограмма QDAWC основана на приведенной в [45] процедуре QAWC.

Пример. Вычисляется интеграл

$$\int_{-1}^{5} \frac{1}{x(5x^3+6)}\,dx = \frac{\ln(125/631)}{18}.$$

```
program qdawcTest
 use dfimsl
 real(4) :: a, b, c, errabs, errest, error, errrel, exact, f, result
 external f
 a = -1.0; b = 5.0; c = 0.0                ! Границы интегрирования и c
 errabs = 0.0; errrel = 0.001              ! Допускаемые ошибки
 call qdawc(f, a, b, c, errabs, errrel, result, errest)
 exact = alog(125.0 / 631.0) / 18.0
 error = abs(result - exact)
 write(*, 99999) result, exact, errest, error
 99999 format(' Computed =', f8.3, 13x, ' Exact =', f8.3, /, /,                &
       ' Error estimate =', 1pe10.3, 6x, ' Error =', 1pe10.3)
end program qdawcTest

real function f(x)
 real(4) :: x
 f = 1.0 / (5.0 * x**3 + 6.0)
end function f
```

Результат:

```
Computed = -0.090                Exact = -0.090
Error estimate = 2.226E-06       Error = 7.451E-09
```

3.2.10. ПОДПРОГРАММА QDNG (DQDNG)

Вычисляет интеграл гладкой функции, не прибегая к адаптивной схеме. Имеет вызов

CALL QDNG(*f, a, b, errabs, errrel, result, errest*)

Описание параметров см. в табл. 3.3.

Комментарии:

1. При работе с QDNG может возникнуть информационная ошибка типа 4 с кодом 1, означающая, что было выполнено максимально допустимое число шагов. Следовательно, интеграл слишком труден для QDNG.
2. Подпрограмма QDAGS создана для производительных вычислений. Если возникает вышеупомянутая ошибка, используйте QDAGS.
3. См. комментарий 2 в разделе с описанием подпрограммы QDAGS.

Описание:

В подпрограмме QDNG реализована неадаптивная квадратурная схема, основанная на вложенных правилах Патерсона порядка 10, 21, 43 и 87. Порядок точности этих правил соответственно 19, 31, 64 и 130. Подпрограмма QDNG применяет эти правила последовательно до тех пор, пока не достигнута заданная точность или не выполнено последнее правило. Подпрограмма QDNG основана на процедуре QNG приведенной в [45].

Пример. Вычисляется интеграл

$$\int_0^2 xe^x dx = e^2 + 1.$$

```
program qdngTest
 use dfimsl
 real(4) :: a, b, errabs, errest, error, errrel, exact, f, result
 external f
 a = 0.0; b = 2.0                        ! Границы интегрирования
 errabs = 0.0; errrel = 0.001            ! Точность результата
 call qdng(f, a, b, errabs, errrel, result, errest)
 exact = 1.0 + exp(2.0)
 error = abs(result - exact)
 write(*, 99999) result, exact, errest, error
 99999 format(' Computed =', f8.3, 13x, ' Exact =', f8.3, /, /,                    &
       ' Error estimate =', 1pe10.3, 6x, ' Error =', 1pe10.3)
end program qdngTest

real function f(x)
 real(4) :: x
 f = x * exp(x)
end function f
```

Результат:

```
Computed = 8.389                    Exact = 8.389
Error estimate = 5.000E-05          Error = 0.000E+00
```

3.3. ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ ФУНКЦИИ НЕСКОЛЬКИХ ПЕРЕМЕННЫХ

3.3.1. ПОДПРОГРАММА TWODQ (DTWODQ)

Вычисляет двойной повторный интеграл. Имеет вызов

CALL TWODQ(*f, a, b, g, h, errabs, errrel, irule, result, errest*)

Параметры подпрограммы TWODQ:

Пользовательские функции: f, g, h.

Входные: a, b, errabs, errrel, irule.

Выходные: result, errest.

Описание параметров *errabs, errrel, result* и *errest* см. в табл. 3.3.

f - пользовательская функция, интеграл которой вычисляется. Имеет вид $f(x, y)$, где x, y - ее вещественные *входные* параметры.

g - пользовательская функция, оценивающая нижнюю границу внутреннего интеграла. Имеет вид $g(x)$, где x - ее вещественный *входной* параметр.

h - пользовательская функция, оценивающая верхнюю границу внутреннего интеграла. Имеет вид $h(x)$, где x - ее вещественный *входной* параметр.

Функции *f, g* и *h* должны получить атрибут EXTERNAL в вызывающей программе.

a, b - соответственно нижняя и верхняя границы внешнего интеграла.

irule - задает квадратурное правило Гаусса - Кронрода, которое используется:

- с 7-15 точками, если *irule* = 1;
- 10-21 точками, если *irule* = 2;
- 15-31 точками, если *irule* = 3;
- 20-41 точками, если *irule* = 4;
- 25-51 точками, если *irule* = 5;
- 30-61 точками, если *irule* = 6.

Если функция имеет пиковую особенность, то используется *irule* = 1. В случае осциллирующей функции задается *irule* = 6.

Автоматически для решения предоставляется память:

- 2500 байт в случае TWODQ;
- 4500 байт в случае DTWODQ.

Память можно выделить явно, употребив T2ODQ (DT2ODQ):

CALL T2ODQ(*f, a, b, g, h, errabs, errrel, irule, result, errest,* &
maxsub, neval, nsubin, alist, blist, rlist, elist, iord, wk, iwk)

Смысл дополнительных параметров подпрограммы T2ODQ, кроме приведенных ниже, см. в табл. 3.4.

nsubin - число подынтервалов, сгенерированных во внешнем интеграле. Является *выходным*.

alist, blist - векторы размера *maxsub*, содержащие списки из *nsubin* соответственно левых и правых точек подынтервалов внешнего интеграла. Являются *выходными*.

rlist - вектор размера *maxsub*, содержащий приблизительные значения *nsubin* интегралов в подынтервалах, определенных векторами *alist* и *blist* и относящихся только к внутреннему интегралу. Является *выходным*.

wk - вещественный рабочий вектор размера 4*maxsub*, необходимый для оценки внутреннего интеграла.

iwk - целочисленный рабочий вектор размера *maxsub*, необходимый для оценки внутреннего интеграла.

Комментарии:

1. При работе с TWODQ могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	1	Достигнуто максимально допустимое число подынтервалов
3	2	Ошибка округления не позволяет обеспечить заданную точность
3	3	Обнаружена потеря точности

2. См. комментарий 2 в разд. 3..3.2.

Описание:

Подпрограмма TWODQ находит приблизительное значение интеграла

$$\int_a^b \int_{g(x)}^{h(x)} f(x,y)\,dy\,dx.$$

При вычислениях повторно вызывается подпрограмма QDAG. Так же как и в QDAG, доступны несколько настроек. При вычислении интеграла менее гладких функций используются Гаусса - Кронрода пары с меньшими номерами. Для оценки интегралов осциллирующих гладких функций применяются Гаусса - Кронрода пары с большими номерами.

Пример 1. Вычисляется интеграл

$$\int_{0}^{1}\int_{1}^{3} y\cos(x+y^2)dydx.$$

```
program twodqTest1
 use dfimsl
 integer(4) :: irule
 real(4) :: a, b, errabs, errest, errrel, f, g, h, result
 external f, g, h
 a = 0.0; b = 1.0                          ! Границы интегрирования
 errabs = 0.0; errrel = 0.01               ! Точность результата
 irule = 6                                 ! Работаем с осциллирующей функцией
 call twodq(f, a, b, g, h, errabs, errrel, irule, result, errest)
 write(*, 99999) result, errest
 99999 format(' Result =', f8.3, 13x, ' Error estimate = ', 1pe9.3)
end program twodqTest1

real function f(x, y)
 real(4) :: x, y
 f = y * cos(x + y * y)
end function f

real function g(x)
 real(4) :: x
 g = 1.0
end function g

real function h(x)
 real(4) :: x
 h = 3.0
end function h
```

Результат:

```
Result = -0.514 Error estimate = 3.065E-06
```

Пример 2. Вычисляется интеграл

$$\int_{0}^{1}\int_{-2x}^{5x} y\cos(x+y^2)dydx.$$

```
program twodqTest2
! Текст программы полностью совпадает с текстом программы twodqTest1
...
end program twodqTest2
```

```
real function f(x, y)
 real(4) :: x, y
 f = y * cos(x + y * y)
end function f

real function g(x)
 real(4) :: x
 g = -2.0 * x
end function g

real function h(x)
 real(4) :: x
 h = 5.0 * x
end function h
```

Результат:

```
*** WARNING ERROR 2 from TWODQ. Roundoff error has been detected. The
*** requested tolerances cannot be reached.
Computed = -0.083 Error estimate = 2.095E-06
```

3.3.2. ПОДПРОГРАММА QAND (DQAND)

Вычисляет *n*-кратный интеграл функции нескольких переменных. Имеет вызов

CALL QAND(*f, n, a, b, errabs, errrel, maxfcn, result, errest*)

Параметры подпрограммы QAND:

Пользовательская функция: f.

Входные: n, a, b, errabs, errrel, maxfcn.

Выходные: result, errest.

Описание параметров *errabs, errrel, result* и *errest* см. в табл. 3.3.

f - пользовательская функция, интеграл которой вычисляется. Имеет вид $f(n, x)$, где *n* - число независимых переменных, *x* - вектор размера *n*, содержащий независимые переменные. Параметры *n* и *x* являются *входными*.

n - число независимых переменных; не должно превышать 20.

a, b - векторы размера *n*, содержащие соответственно нижние и верхние пределы интегрирования.

maxfcn - максимально допустимое число оценок функции; не должно быть больше 256^n.

Комментарии:

1. При работе с QAND могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	1	Значение *maxfcn* больше чем 256^n
4	2	Выполнено максимальное число оценок функции, но сходимость не достигнута

2. См. комментарий 2 в разд. 3.2.2.

Описание:

Подпрограмма QAND находит приблизительное значение n-мерного интеграла

$$\int_{a_1}^{b_1}\cdots\int_{a_n}^{b_n} f(x_1,\ldots,x_n)dx_n\ldots dx_1 .$$

Приближение интеграла достигается в результате последовательного применения формул произведения Гаусса. Первоначально интеграл оценивается двухточечной формулой тензорного произведения в каждом измерении. Затем для $i = 1, \ldots, n$ подпрограмма вычисляет новую оценку, удваивая число точек в i-м измерении, возвращаясь к прежнему значению, если новая оценка неудовлетворительна. Процесс завершается, если не наблюдается улучшения оценки интеграла, или число точек Гаусса в одном измерении превысило 256, или число оценок функции превышает *maxfcn*.

Пример. Вычисляется приближенное значение интеграла функции

$$e^{-(x_1^2+x_2^2+x_3^2)}$$

в расширяющемся кубе. Известно, что

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-(x_1^2+x_2^2+x_3^2)} = \pi^{3/2} .$$

```
program qandTest
 use dfimsl
 integer(4) :: i, j, maxfcn, n
 real(4) :: a(3), b(3), con, errabs, errest, errrel, f, result
 external f
 n = 3; maxfcn = 100000
 errabs = 0.0001; errrel = 0.001          ! Точность результата
 do i = 1, 6
  con = float(i) / 2.0
```

```
 ! По мере приближения con к ∞ ответ стремится к π^3/2
 a(1:3) = -con; b(1:3) = con              ! Пределы интегрирования
 call qand(f, n, a, b, errabs, errrel, maxfcn, result, errest)
 write(*, 99999) con, result, errest
 end do
 99999 format(1x, 'For con = ', f4.1, ', result = ', f7.3, ' with error estimate ', 1pe10.3)
end program qandTest

real function f(n, x)
integer(4) :: n
real(4) :: x(n)
 f = exp(-(x(1) * x(1) + x(2) * x(2) + x(3) * x(3)))
end function f
```

Результат:

```
For con = 0.5, result = 0.785 with error estimate 3.934E-06
For con = 1.0, result = 3.332 with error estimate 2.100E-03
For con = 1.5, result = 5.021 with error estimate 1.192E-05
For con = 2.0, result = 5.491 with error estimate 2.422E-04
For con = 2.5, result = 5.561 with error estimate 4.231E-03
For con = 3.0, result = 5.568 with error estimate 2.584E-04
```

3.4. ПРАВИЛА ГАУССА И ТРЕХЭЛЕМЕНТНЫЕ РЕКУРРЕНТНЫЕ СООТНОШЕНИЯ

3.4.1. ПЕРЕЧЕНЬ И ПАРАМЕТРЫ ПОДПРОГРАММ

Перечень подпрограмм см. в табл. 3.5, их параметры - в табл. 3.6.

Таблица 3.5. Подпрограммы, вычисляющие правила Гаусса и трехэлементные рекуррентные соотношения

Подпрограмма	*Назначение*
GQRUL	Вычисляет квадратурное правило Гаусса, Гаусса - Радау или Гаусса - Лобатто с различными весовыми функциями
GQRCF	Вычисляет квадратурное правило Гаусса, Гаусса - Радау или Гаусса - Лобатто с данными рекуррентными коэффициентами для ортогональных многочленов с различными весовыми функциями
RECCF	Вычисляет рекуррентные коэффициенты для различных ортогональных многочленов
RECQR	Вычисляет рекуррентные коэффициенты ортогонального многочлена заданного квадратурным правилом

Таблица 3.6. Параметры подпрограмм из табл. 3.5

<table>
<tr><th>Имя</th><th>Смысл/вид</th><th>Тип</th></tr>
<tr><td>alpha</td><td>Параметр, используемый в весовой функции с iweigh = 5 или iweigh = 6; в других случаях игнорируется / входной</td><td>REAL(4) или REAL(8)</td></tr>
<tr><td>b</td><td>Вектор размера n, содержащий рекуррентные коэффициенты / входной в подпрограмме GQRCF; выходной в подпрограммах RECCF и RECQR</td><td>То же</td></tr>
<tr><td>beta</td><td>Параметр, используемый в весовой функции с iweigh = 5; в других случаях игнорируется / входной</td><td>"</td></tr>
<tr><td>c</td><td>Вектор размера n, содержащий рекуррентные коэффициенты / входной в подпрограмме GQRCF; выходной в подпрограммах RECCF и RECQR</td><td>"</td></tr>
<tr><td>iweigh</td><td>Индекс весовой функции w(x) / входной; выбирается из следующей таблицы:
<table>
<tr><th>iweigh</th><th>w(x)</th><th>Интервал</th><th>Весовая функция</th></tr>
<tr><td>1</td><td>1</td><td>(-1, 1)</td><td>Лежандра</td></tr>
<tr><td>2</td><td>$1/\sqrt{1-x^2}$</td><td>(-1, 1)</td><td>Чебышева 1-го рода</td></tr>
<tr><td>3</td><td>$\sqrt{1-x^2}$</td><td>(-1, 1)</td><td>Чебышева 2-го рода</td></tr>
<tr><td>4</td><td>e^{-x^2}</td><td>(-∞, ∞)</td><td>Эрмита</td></tr>
<tr><td>5</td><td>$(1+x)^{\alpha}(1-x)^{\beta}$</td><td>(-1, 1)</td><td>Якоби</td></tr>
<tr><td>6</td><td>$e^{-x}x^{\alpha}$</td><td>(0, ∞)</td><td>Обобщенная Лагерра</td></tr>
<tr><td>7</td><td>1/cosh(x)</td><td>(-∞, ∞)</td><td>Гиперболический косинус</td></tr>
</table></td><td>INTEGER(4)</td></tr>
<tr><td>n</td><td>Число квадратурных точек / входной</td><td>"</td></tr>
<tr><td>nfix</td><td>Число фиксированных квадратурных точек; nfix = 0, или 1, или 2. Обычно для квадратурных правил Гаусса полагают nfix = 0 / входной</td><td>"</td></tr>
<tr><td>qx</td><td>Вектор размера n, содержащий квадратурные точки / выходной</td><td>REAL(4) или REAL(8)</td></tr>
<tr><td>qxfix</td><td>Вектор размера nfix (игнорируется, если nfix = 0), содержащий фиксированные квадратурные точки / входной</td><td>То же</td></tr>
<tr><td>qw</td><td>Вектор размера n, содержащий квадратурные веса</td><td>"</td></tr>
</table>

3.4.2. ПОДПРОГРАММА GQRUL (DGQRUL)

Вычисляет квадратурное правило Гаусса, Гаусса - Радау или Гаусса - Лобатто с различными весовыми функциями. Имеет вызов

CALL GQRUL(*n, iweigh, alpha, beta, nfix, qxfix, qx, qw*)

Описание параметров подпрограммы GQRUL см. в табл. 3.6.

Комментарии:

1. Квадратуры Гаусса хороши, если функция $f(x)$ ведет себя как многочлен. Также гауссовы квадратуры (с подходящими весовыми функциями) полезны при работе с бесконечным интервалом, поскольку иные методы часто на таком интервале не работают.
2. Весовая функция $1/\cosh(x)$ ведет как многочлен вблизи нуля и как $e^{|x|}$ вдали от него.

Описание:

Процедура GQRUL вырабатывает точки и веса по квадратурным формулам Гаусса, Гаусса - Радау или Гаусса - Лобатто. Одна или две точки могут быть фиксированными, причем они могут не совпадать с границами отрезка интегрирования. Подпрограмма GQRUL является модификацией процедуры GAUSSQUADRULE, приведенной в [31].

В простом случае, когда *nfix* = 0, подпрограмма возвращает точки в векторе $x = qx$ и веса в векторе $w = qw$, такие, что

$$\int_a^b f(x)w(x)dx \approx \sum_{i=1}^{n} f(x_i)w_i$$

для всех функций f, являющихся многочленами, степень которых меньше чем $2n$.

Если *nfix* = 1, то один из элементов x_i равняется первому элементу *qxfix*. Если *nfix* = 2, то два элемента вектора x равны элементам *qxfix*(1:2). В общем случае точность приведенной квадратурной формулы падает при увеличении *nfix*. Квадратурное правило вычисляет интеграл всех функций f, являющихся многочленами, степень которых меньше, чем $2n$ - *nfix*.

Пример 1. Первоначально получаем классическую Гаусса - Лежандра квадратурную формулу, точную для многочленов, степень которых меньше $2n$, а затем применяем ее в ситуации, когда $n = 6$, к функции x^8 на отрезке [-1, 1]. Это квадратурное правило точно для многочленов, степень которых меньше 12.

Пример 2. Дополнительно к сформулированной в примере 1 задаче требуется, чтобы в квадратурную формулу были включены обе концевые точки. Новые квадратурные формулы применяем к функции x^8 на отрезке [-1, 1]. Это квадратурное правило точно для многочленов, степень которых меньше 10.

```
program gqrulTest
 use dfimsl, nouse => beta                ! Чтобы преодолеть конфликт имен
 integer(4), parameter :: n = 6
 integer(4) :: iweigh, nfix, nout
 real(4) :: alpha, answer, beta, qw(n), qx(n), qxfix(2)
 iweigh = 1; alpha = 0.0; beta = 0.0
 ! Инициализация nfix в примере 1; вектор qxfix игнорируется
 nfix = 0
 ! Инициализация nfix и вектора qxfix в примере 2. Чтобы запустить пример 2,
 ! нужно снять символы комментария перед следующими тремя операторами:
 ! nfix = 2
 ! qxfix(1) = -1.0
 ! qxfix(2) = 1.0
 ! Получаем точки и веса из GQRUL
 call gqrul(n, iweigh, alpha, beta, nfix, qxfix, qx, qw)
 ! Выводим результат, полученный в GQRUL
 write(*, 99998) (i, qx(i), i, qw(i), i = 1, n)
 99998 format(6(6x, 'qx(', i1, ') = ', f8.4, 7x, 'qw(', i1, ') = ', f8.5, /))
 ! Оцениваем интеграл по полученным точкам и весам
 answer = sum(qx**8*qw)
 write(*, 99999) answer
 99999 format (/, ' The quadrature result making use of these points and weights is ', 1pe10.4)
end program gqrulTest
```

Результат:

Пример 1		Пример 2	
qx(1) = -0.9325	qw(1) = 0.17132	qx(1) = -1.0000	qw(1) = 0.06667
qx(2) = -0.6612	qw(2) = 0.36076	qx(2) = -0.7651	qw(2) = 0.37847
qx(3) = -0.2386	qw(3) = 0.46791	qx(3) = -0.2852	qw(3) = 0.55486
qx(4) = 0.2386	qw(4) = 0.46791	qx(4) = 0.2852	qw(4) = 0.55486
qx(5) = 0.6612	qw(5) = 0.36076	qx(5) = 0.7651	qw(5) = 0.37847
qx(6) = 0.9325	qw(6) = 0.17132	qx(6) = 1.0000	qw(6) = 0.06667

The quadrature result making use of these points and weights is 2.2222E-01.

3.4.3. ПОДПРОГРАММА GQRCF (DGQRCF)

Вычисляет квадратурное правило Гаусса, Гаусса - Радау или Гаусса - Лобатто с данными рекуррентными коэффициентами для ортогональных многочленов с различными весовыми функциями. Имеет вызов

CALL GQRCF(*n, b, c, nfix, qxfix, qx, qw*)

Параметры подпрограммы GQRCF:

Входные: n, b, c, nfix, qxfix.

Выходные: qx, qw.

Описание параметров см. в табл. 3.6.

Комментарии:

1. При работе с GQRCF может возникнуть информационная ошибка типа 4 с кодом 1, означающая, что за 100 итераций не достигнуто сходимости.
2. См. комментарий 1 в предшествующем разделе.
3. Рекуррентные коэффициенты bi и ci, содержащиеся соответственно в векторах b и c, определяют ортогональный многочлен через соотношение

$$p_i = (x - b_i)p_{i-1}(x) - c_i p_{i-2}(x),\ i = 1, 2, \ldots n,$$

где $p_0 = 1$, $p_{-1} = 0$. Каждый элемент вектора *c* должен быть больше нуля.

Описание:

Подпрограмма GQRCF является модификацией процедуры GAUSSQUADRULE, приведенной в [31]. Она вызывается после получения в подпрограмме RECCF рекуррентных коэффициентов. Параметр *iweigh* подпрограммы RECCF задает вид весовой функции. Получая рекуррентные коэффициенты, подпрограмма GQRCF, когда *nfix* = 0, возвращает точки в векторе $x = qx$ и веса в векторе $w = qw$, такие, что

$$\int_a^b f(x)w(x)dx \approx \sum_{i=1}^{n} f(x_i)w_i$$

для всех функций *f*, являющихся многочленами, степень которых меньше, чем $2n$. Весовая функция $w(x)$ нормализуется так, что

$$\int_a^b w(x)dx = c_1 .$$

Далее об употреблении *nfix* см. в предшествующем разделе.

Пример. Вычисляется квадратурное правило Гаусса с $n = 6$ с весами Чебышева 1-го рода и коэффициентами рекуррентного соотношения, возвращаемыми подпрограммой RECCF.

```
program gqrcfTest
 use dfimsl, nouse => beta                    ! Чтобы преодолеть конфликт имен
 integer(4), parameter :: n = 6
 integer(4) :: iweigh, nfix
 real(4) :: b(n), c(n), qw(n), qx(n), qxfix(2)
 iweigh = 1; alpha = 0.0; beta = 0.0
 ! Подпрограмма RECCF вернет коэффициенты рекуррентного соотношения для
 ! многочленов Чебышева 1-го рода
 iweigh = 1
 alpha = 0.0; beta = 0.0
 call reccf(n, iweigh, alpha, beta, b, c)
 nfix = 0
 ! Подпрограмма GQRCF вернет квадратурное правило,
 ! используя полученные выше коэффициенты рекуррентного соотношения
 call gqrcf(n, b, c, nfix, qxfix, qx, qw)
 ! Выводим результат, полученный в GQRCF
 write(*, 99998) (i, qx(i), i, qw(i), i = 1, n)
 99998 format(6(6x, 'qx(', i1, ') = ', f8.4, 7x, 'qw(', i1, ') = ', f8.5, /))
end program gqrcfTest
```

Результат:

```
 qx(1) = -0.9325     qw(1) = 0.17132
 qx(2) = -0.6612     qw(2) = 0.36076
 qx(3) = -0.2386     qw(3) = 0.46791
 qx(4) = 0.2386      qw(4) = 0.46791
 qx(5) = 0.6612      qw(5) = 0.36076
 qx(6) = 0.9325      qw(6) = 0.17132
```

3.4.4. ПОДПРОГРАММА RECCF (DRECCF)

Вычисляет рекуррентные коэффициенты для различных ортогональных многочленов. Имеет вызов

CALL RECCF(*n, iweigh, alpha, beta, b, c*)

Описание параметров подпрограммы RECCF, кроме *входного* параметра *n*, см. в табл. 3.6.

n - число рекуррентных коэффициентов.

См. *комментарий 3* в предшествующем разделе.

Пример. Вычисляются рекуррентные коэффициенты для первых шести многочленов Лежандра, Чебышева 1-го рода и Лагерра.

```
program reccfTest
 use dfimsl, nouse => beta                    ! Чтобы преодолеть конфликт имен
 integer(4), parameter :: n = 6
 integer(4) :: i, iweigh
 real(4) :: alpha, b(n), beta, c(n)
 alpha = 0.0; beta = 0.0
 iweigh = 1
 call reccf(n, iweigh, alpha, beta, b, c)
 write(*, "(1x, 'Legendre')")
 write(*, 99999) (i, b(i), i, c(i), i = 1, n)
 iweigh = 2
 call reccf(n, iweigh, alpha, beta, b, c)
 write(*, "(/, 1x, 'Chebyshev, first kind')")
 write(*, 99999) (i, b(i), i, c(i), i = 1, n)
 iweigh = 6
 call reccf(n, iweigh, alpha, beta, b, c)
 write(*, "(/, 1x, 'Laguerre')")
 write(*, 99999) (i, b(i), i, c(i), i = 1, n)
 99999 format (6(6x, 'b(', i1, ') = ', f8.4, 7x, 'c(', i1, ') = ', f8.5,/))
end program reccfTest
```

Результат:

```
Legendre
b(1) = 0.0000      c(1) = 2.00000
b(2) = 0.0000      c(2) = 0.33333
b(3) = 0.0000      c(3) = 0.26667
b(4) = 0.0000      c(4) = 0.25714
b(5) = 0.0000      c(5) = 0.25397
b(6) = 0.0000      c(6) = 0.25253

Chebyshev, first kind
b(1) = 0.0000      c(1) = 3.14159
b(2) = 0.0000      c(2) = 0.50000
b(3) = 0.0000      c(3) = 0.25000
b(4) = 0.0000      c(4) = 0.25000
b(5) = 0.0000      c(5) = 0.25000
b(6) = 0.0000      c(6) = 0.25000
```

```
Laguerre
b(1) = 1.0000      c(1) = 1.00000
b(2) = 3.0000      c(2) = 1.00000
b(3) = 5.0000      c(3) = 4.00000
b(4) = 7.0000      c(4) = 9.00000
b(5) = 9.0000      c(5) = 16.00000
b(6) = 11.0000     c(6) = 25.00000
```

3.4.5. ПОДПРОГРАММА RECQR (DRECQR)

Вычисляет рекуррентные коэффициенты ортогонального многочлена, заданного квадратурным правилом. Имеет вызов

CALL RECQR(*n, qx, qw, nterm, b, c*)

Описание параметров подпрограммы RECQR, кроме *nterm*, см. в табл. 3.6.

nterm - число рекуррентных коэффициентов; *nterm* ≤ *n*.

См. *комментарий 3* в разд. 3.4.3.

Описание:

Подпрограмма RECQR возвращает рекуррентные коэффициенты для ортогональных многочленов, заданных точками и весами квадратурной формулы Гаусса. Подпрограмма является обратной по отношению к подпрограмме GQRCF.

Пример. Первоначально генерируются рекуррентные коэффициенты; затем подпрограммой GQRCF вычисляется квадратурная формула; в довершение всего векторы, содержащие квадратурные точки и веса, передаются подпрограмме RECQR, которая восстанавливает рекуррентные коэффициенты.

```
program recqrTest
 use dfimsl
 integer(4), parameter :: n = 5
 integer(4) :: i, j, nfix, nterm
 real(4) :: b(n), c(n), float, qw(n), qx(n), qxfix(2)
 nfix = 0
 ! Формируем векторы с рекуррентными коэффициентами
 do j = 1, n; b(j) = float(j); c(j) = float(j) / 2.0; end do
 write(*, "(1x, 'Original recurrence coefficients')")
 write(*, 99996) (i, b(i), i, c(i), i = 1, n)
 99996 format (5(6x, 'b(', i1, ') = ', f8.4, 7x, 'c(', i1, ') = ', f8.5,/))
 ! Вызываем GQRCF и вычисляем квадратурное правило
```

```
! для сгенерированных выше рекуррентных коэффициентов
call gqrcf(n, b, c, nfix, qxfix, qx, qw)
write(*, "(/, 1x, 'Quadrature rule from the recurrence coefficients')")
write(*, 99998) (i, qx(i), i, qw(i), i = 1, n)
99998 format (5(6x, 'b(', i1, ') = ', f8.4, 7x, ' qw(', i1, ') = ', f8.5,/))
! Вызываем RECQR и восстанавливаем
! первоначальные рекуррентные коэффициенты
nterm = n
call recqr(n, qx, qw, nterm, b, c)
write(*, "(/, 1x, 'Recurrence coefficients determined by RECQR')")
write(*, 99996) (i, b(i), i, c(i), i = 1, n)
end program recqrTest
```

Результат:

```
Original recurrence coefficients
b(1) = 1.0000      c(1) = 0.50000
b(2) = 2.0000      c(2) = 1.00000
b(3) = 3.0000      c(3) = 1.50000
b(4) = 4.0000      c(4) = 2.00000
b(5) = 5.0000      c(5) = 2.50000

Quadrature rule from the recurrence coefficients
qx(1) = 0.1525     qw(1) = 0.25328
qx(2) = 1.4237     qw(2) = 0.17172
qx(3) = 2.7211     qw(3) = 0.06698
qx(4) = 4.2856     qw(4) = 0.00790
qx(5) = 6.4171     qw(5) = 0.00012

Recurrence coefficients determined by RECQR
b(1) = 1.0000      c(1) = 0.50000
b(2) = 2.0000      c(2) = 1.00000
b(3) = 3.0000      c(3) = 1.50000
b(4) = 4.0000      c(4) = 2.00000
b(5) = 5.0000      c(5) = 2.50000
```

3.4.6. ПОДПРОГРАММА FQRUL (DFQRUL)

Вычисляет квадратурные правила Фейера с различными весовыми функциями. Имеет вызов

CALL FQRUL(*n, a, b, iweigh, alpha, beta, qx, qw*)

Описание параметров подпрограммы RECQR, кроме *входных* параметров *iweigh, alpha* и *beta*, см. в табл. 3.6.

iweigh - индекс весовой функции $w(x)$; выбирается из следующей таблицы:

iweigh	$w(x)$
1	1
2	$1/(x - \alpha)$
3	$(b - x)^{\alpha}(x - a)^{\beta}$
4	$(b - x)^{\alpha}(x - a)^{\beta}\ln(x - a)$
5	$(b - x)^{\alpha}(x - a)^{\beta}\ln(b - x)$

alpha - параметр, используемый в весовой функции с *iweigh* = 2-4. Игнорируется, если *iweigh* = 1. Если *iweigh* = 2, то должен удовлетворять неравенству $a < alpha < b$. Если *iweigh* = 3, 4 или 5, то должен быть больше, чем -1.0.

beta - параметр, используемый в весовой функции с *iweigh* = 3-4. В других случаях игнорируется; должен быть больше, чем -1.0.

Комментарии:

1. Подпрограмма возвращает точки в векторе $x = qx$ и веса в векторе $w = qw$, такие, что на отрезке $[a, b]$

$$\int_a^b f(x)w(x)dx \approx \sum_{i=1}^{n} f(qx_i)qw_i .$$

2. Подпрограмма FQRUL использует быстрые преобразования Фурье, поэтому она наиболее эффективна, когда n является произведением небольших простых сомножителей.

Описание:

Подпрограмма FQRUL возвращает веса и точки квадратурного правила Фейера. Вариант применения квадратурных формул Фейера в качестве промежуточных звеньев сложных ситуаций см. в [28].

Правила Фейера основываются на интерполяции многочленов. Для простоты изложения рассмотрим случай, когда интегрирование выполняется на отрезке [-1, 1]. Первоначально выбираются классические абсциссы (в нашем случае - это точки Гаусса для весовой функции Чебышева $(1 - x^2)^{-1/2}$), а затем получается квадратурное правило для различных весов. Квадратурное правило ищется в виде

$$q(f) \approx \sum_{j=1}^{n} f(x_j)w_j \ , j = 1, \ldots, n,$$

где x_j - корень многочлена Чебышева 1-го рода $T_n(x) = \cos(n\arccos x)$ степени n.

Веса в квадратурном правиле q выбираются такими, что для всех многочленов p степени, меньшей n,

$$q(p) \approx \sum_{j=1}^{n} p(x_j)w_j = \int_{-1}^{1} p(x)w(x)dx$$

для некоторой весовой функции $w(x)$.

Правила Фейера полезны, поскольку могут быть вычислены четвертьбыстрыми преобразованиями Фурье, т. е. с высокой производительностью.

Если p в последней формуле заменить на T_l, мы получим систему линейных уравнений

$$q(T_l) \approx \sum_{j=1}^{n} T_l(x_j)w_j = \int_{-1}^{1} T_l(x)w(x)dx,\ j = 1, \ldots, n - 1,$$

с неизвестными весами w_j. Систему можно упростить, если заметить, что

$$x_j = \cos\frac{(2j-1)\pi}{2n},\ j = 1, \ldots, n,$$

и, следовательно,

$$\int_{-1}^{1} T_l(x)w(x)dx = \sum_{j=1}^{n} T_l(x_j)w_j = \sum_{j=1}^{n} w_j \cos\frac{l(2j-1)\pi}{2n}.$$

Последнее выражение является четвертьбыстрым прямым косинус-преобразованием Фурье последовательности $w(1{:}n)$. Такое преобразование реализуется подпрограммой QCOSF. Обратное преобразование выполняется подпрограммой QCOSB, посредством которой и происходит вычисление интеграла в последнем выражении. Детали см. в [22, с. 84-86] и [27, с. 259].

Пример. Вычисляется квадратурное правило Фейера с использованием 10, 100 и 200 точек. По этим правилам находится (с нарастающей точностью) значение интеграла

$$\int_{0}^{1} x\sin(41\pi x^2)dx = \frac{1}{41\pi}.$$

```
program fqrulTest
 use dfimsl, nouse => beta                ! Чтобы преодолеть конфликт имен
 integer(4), parameter :: nmax = 200
 integer(4) :: i, iweigh, k, n, nfix
 real(4) :: a, alpha, answer, b, beta, qw(nmax), qx(nmax), x, pi, error
 pi = const('pi')                         ! Число π
```

```
 do k = 1, 3
  if(k == 1) n = 10; if(k == 2) n = 100; if(k == 3) n = 200
  a = 0.0; b = 1.0; iweigh = 1
  alpha = 0.0; beta = 0.0; nfix = 0
  ! Получаем квадратурные точки и веса
  call fqrul(n, a, b, iweigh, alpha, beta, qx, qw)
  ! Оцениваем интеграл по полученным квадратурным точкам и весам
  answer = sum(f(qx) * qw)
  error = abs(answer - 1.0 / (41.0 * pi))
  write(*, 9) n, answer, error
 end do
 9 format(/, 1x, 'When n = ', i3, ', the quadrature result making use of these points ',    &
       /, ' and weights is ', 1pe11.4, ', with error ', 1pe9.2)

contains

 elemental real(4) function f(x)
  real(4), intent(in) :: x
  f = x * sin(41.0 * pi * x**2)
 end function f
end program fqrulTest
```

Результат:

```
When n = 10, the quadrature result making use of these points
and weights is -1.6523E-01, with error 1.73E-01
When n = 100, the quadrature result making use of these points
and weights is 7.7637E-03, with error 5.25E-08
When n = 200, the quadrature result making use of these points
and weights is 7.7638E-03, with error 1.68E-08
```

3.5.ЧИСЛЕННОЕ ДИФФЕРЕНЦИРОВАНИЕ. ФУНКЦИЯ DERIV (DDERIV)

Выполняет оценку первой, второй или третьей производной предоставленной пользователем функции. Имеет вызов

result = DERIV(*fcn, korder, x, bgstep, tol*)

Параметры функции DERIV:

Пользовательская функция: fcn.

Входные: korder, x, bgstep, tol.

Результирующая переменная: DERIV.

fcn - пользовательская функция, производная которой вычисляется в точке *x*. Имеет вид *fcn*(*x*), где *x* - *входная* независимая переменная. Функция *fcn* должна получить атрибут EXTERNAL в вызывающей программе.

korder - порядок вычисляемой производной; может быть равен 1, 2 или 3.

x - точка, в которой оценивается производная.

bgstep - начальное значение, употребляемое при вычислении размера отрезка, определяемого как [*x* - 4*bgstep*, *x* + 4*bgstep*] и используемого для вычисления производной. Значение *bgstep* должно быть больше нуля.

tol - относительная ошибка, задающая желаемую точность вычисления производной.

DERIV - результирующая переменная, содержащая оценку первой (*korder* = 1), второй (*korder* = 2) или третьей (*korder* = 3) производной функции *fcn* в точке *x*.

Комментарии:

1. При работе с QDAGS могут возникать следующие информационные ошибки:

Тип	*Код*	*Описание*
3	2	Ошибка округления не позволяет обеспечить заданную точность. Увеличьте точность или *bgstep*
4	5	Невозможно получить заданный допуск *tol* оценки производной. Увеличьте точность вычислений, *tol* и/или измените *bgstep*

1. Сходимость достигается, когда (2/3)|*d*2 - *d*1| < *tol* для двух последовательных оценок производной *d*1 и *d*2.
2. Начальный шаг *bgstep* должен быть достаточно мал, чтобы функция *fcn* была определенной и приемлемо гладкой на отрезке [*x* - 4*bgstep*, *x* + 4*bgstep*], и достаточно большим, чтобы избежать ошибок округления.

Описание:

Функция DERIV для оценки производной вычисляет на отрезке [*x* - 4*bgstep*, *x* + 4*bgstep*] интерполяционный сплайн, выполняя затем его дифференцирование в точке *x*.

Пример 1. Оценивается первая производная функции $f(x) = -2\sin(3x/2)$ в точке $x = 2$.

```
program derivTest1
 use dfimsl
 integer(4) :: korder, ncount
 real(4) :: bgstep, derv, fcn, tol, x
```

```
 external fcn
 x = 2.0; bgstep = 0.2
 tol = 0.01; korder = 1; ncount = 1
 derv = deriv(fcn, korder, x, bgstep, tol)
 write(*, "(/, 1x, 'First derivative of fcn is ', 1pe10.3)") derv
end program derivTest1

real function fcn(x)
 real(4) :: x
 fcn = -2.0 * sin(1.5 * x)
end function fcn
```

Результат:

```
First derivative of fcn is 2.970E+00
```

Пример 2. Выполняется попытка найти с одинарной точностью приблизительное значений третьей производной функции $f(x) = 2x^4 + 3x$ в точке $x = 0.75$. Хотя поведение функции в окрестности точки $x = 0.75$ хорошее, вычисление производной с одинарной точностью затруднительно. Проблема решается за счет перехода к двойной точности.

```
program derivtest2
use dfimsl
integer(4) :: korder, nout
real(4) :: bgstep, derv, fcn, tol, x
real(8) :: dbgste, dderv, dfcn, dtol, dx
external dfcn, fcn
 ! Будем продолжать вычисления при возникновении
 ! в процедуре завершающей ошибки
 call erset(0, -1, 0)
 x = 0.75; bgstep = 0.1
 tol = 0.01; korder = 3
 ! При одинарной точности на 32-разрядной машине
 ! следующий вызов приведет к ошибке
 derv = deriv(fcn, korder, x, bgstep, tol)
 ! Двойная точность позволяет получить хороший результат
 dx = 0.75d0; dbgste = 0.1d0
 dtol = 0.01d0; korder = 3
 dderv = dderiv(dfcn, korder, dx, dbgste, dtol)
 write(*, "(/, 1x, 'The third derivative of dfcn is ', 1pd10.4)") dderv
end program derivtest2

real(4) function fcn(x)
 real(4) :: x
 fcn = 2.0 * x**4 + 3.0 * x
```

```
end function fcn

real(8) function dfcn(x)
 real(8) :: x
 dfcn = 2.0d0 * x**4 + 3.0d0 * x
end function dfcn
```

Результат:

```
*** FATAL ERROR 1 from DERIV. Unable to achieve desired tolerance.
*** Increase precision, increase TOL = 1.000000E-02 and/or change
*** BGSTEP = 1.000000E-01. If this error continues the function
*** may not have a derivative at X = 7.500000E-01

The third derivative of dfcn is 3.6000D+01
```

4. РЕШЕНИЕ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

4.1. НЕКОТОРЫЕ СВЕДЕНИЯ О ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЯХ

4.1.1. БАЗОВЫЕ ПОНЯТИЯ

Дифференциальное уравнение - это уравнение, в которое неизвестная функция или вектор-функция входит под знаком производной или дифференциала.

Дифференциальное уравнение с одной независимой переменной называется *обыкновенным дифференциальным уравнением*. Например, таким является уравнение радиоактивного распада

$$\frac{d\ x}{d\ t} = -kx, \tag{4.1}$$

где k - постоянная распада; x - количество неразложившегося вещества в момент времени t; скорость распада dx/dt пропорциональна количеству распадающегося вещества. В уравнении (4.1) переменная t является *независимой*, а x - *зависимой*.

Примером обыкновенного дифференциального уравнения с вектор-функцией может служить система

$$\begin{array}{lllllllll}
y_1'(t) & = & a_{11}y_1(t) & + & a_{12}y_2(t) & + & \ldots & + & a_{1n}y_n(t); \\
y_2'(t) & = & a_{21}y_1(t) & + & a_{22}y_2(t) & + & \ldots & + & a_{2n}y_n(t); \\
\ldots\ldots & \ldots & \ldots\ldots & \ldots & \ldots\ldots & \ldots & \ldots & \ldots & \ldots\ldots \\
y_n'(t) & = & a_{n1}y_1(t) & + & a_{n2}y_2(t) & + & \ldots & + & a_{nn}y_n(t).
\end{array}$$

Матричная форма приведенной системы:

$$\begin{pmatrix} y_1'(t) \\ y_2'(t) \\ \ldots \\ y_n'(t) \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix} \begin{pmatrix} y_1(t) \\ y_2(t) \\ \ldots \\ y_n(t) \end{pmatrix},$$

или:

$$y'(t) = A\,y(t). \tag{4.2}$$

Приведенная система служит простейшим примером системы дифференциальных уравнений первого порядка с постоянными коэффициентами.

В общем виде обыкновенное дифференциальное уравнение первого порядка выглядит так:

$$y'(t) = \frac{dy}{dt} = f(t, y(t)),$$

Дифференциальное уравнение, в котором число независимых переменных больше единицы, называется *дифференциальным уравнением в частных производных*. Например, уравнение теплопроводности с постоянными коэффициентом теплопроводности k, плотностью ρ и теплоемкостью единицы длины $c\rho$, описывающее процесс распространения тепла в одномерном стержне $0 < x < l$, имеет вид

$$\frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial x^2} + f,$$

где $u = u(x, t)$ - температура в точке x стержня в момент времени t; $a^2 = k/(c\rho)$; $f = f_0/(c\rho)$; $f_0 = f_0(x, t)$ - плотность тепловых источников.

Порядком дифференциального уравнения называется максимальный порядок входящей в уравнение производной или дифференциала.

Решением дифференциального уравнения называется функция, которая при подстановке в уравнение обращает его в тождество. Процесс нахождения решений дифференциального уравнения принято называть *интегрированием* этого уравнения. График решения дифференциального уравнения называют *интегральной кривой*. В общем случае, когда не заданы начальные условия, решением дифференциального уравнения является семейство функций. Чтобы выделить из семейства функций одно конкретное решение, задают начальное условие

$$y(t_0) = y_0,$$

в котором t_0 - фиксированное значение аргумента t, а y_0 - величина, называемая начальным значением. Например, решением уравнения (4.1) с заданными начальными условиями, т. е. когда известно количество распадающегося вещества x_0 в некоторый начальный момент времени t_0, является функция

$$x = x_0 e^{-k(t-t_0)}.$$

Решением уравнения (4.2) является вектор-функция

$$y(t) = \sum_{i=1}^{n} c_i e^{\lambda_i (t-t_0)} e_i,$$

где $c = (c_1, c_2, ..., c_n)^T = P^{-1} y(t_0)$; λ_i и e_i - соответственно собственное значение и собственный вектор матрицы A. Столбцами матрицы P являются собственные векторы $e_1, e_2, ..., e_n$.

Системы уравнений, включающих производные с одной независимой переменной и другие зависимые переменные и имеющие вид

$$g(t, y, y') = (g_1(t, y, y'), ..., g_n(t, y, y')) = 0,$$

называются *дифференциальными алгебраическими уравнениями*.

В зависимости от дополнительных условий, налагаемых на решение, различают *задачу Коши* и *краевую задачу*. В библиотеке IMSL имеются процедуры для решения обоих видов задач.

4.1.2. ЗАДАЧА КОШИ, ИЛИ НАЧАЛЬНАЯ ЗАДАЧА

Многие явления с известной степенью точности можно описать дифференциальными уравнениями. Довольно часто приходится решать задачи, в которых известно состояние системы в момент времени t_0 и надо предсказать ее поведение при $t > t_0$. Подобного рода задачи, в которых решаются дифференциальные уравнения с начальными условиями, называют *задачами Коши* или *начальными задачами*.

Таким образом, задача Коши описывает развитие процессов во времени. Понимание этого обстоятельства полезно для выбора методов решения и критериев оценки его качества.

Замечание. В роли t может выступать иная, например пространственная, переменная.

Для численного решения задачи Коши используются разнообразные методы дискретизации, подразумевающие переход от непрерывной задачи к дискретной. В частности, IMSL применяет для решения задачи Коши методы Рунге - Кутты, Адамса, Гира и их модификации.

4.1.3. ДВУХТОЧЕЧНАЯ КРАЕВАЯ ЗАДАЧА

Двухточечная краевая задача - это задача отыскания решения обыкновенного дифференциального уравнения или системы обыкновенных дифференциальных уравнений на отрезке $a \le x \le b$, когда дополнительные условия на решение налагаются в точках a и b - краях отрезка $[a, b]$.

Решать краевую задачу сложнее, чем задачу Коши. Численные методы ее решения основаны на той же, что и в случае задачи Коши, идее, предполагающей замену исходной задачи ее дискретным аналогом. Для решения краевой задачи в подпрограммах BVPFD (DBVPFD) и BVPMS (DBVPMS) библиотеки IMSL реализованы модификации методов конечных разностей, прямых и многократной пристрелки. Заметим, что подпрограмма BVPMS (DBVPMS) показывает хорошие результаты при решении плохо обусловленных задач.

4.1.4. ДИФФЕРЕНЦИАЛЬНЫЕ УРАВНЕНИЯ ВЫСОКОГО ПОРЯДКА

Многие процессы моделируются дифференциальными уравнениями второго и более высоких порядков. Обыкновенное дифференциальное уравнение n-го второго порядка - это уравнение вида

$$y^{(n)} = f(t, y, y', ..., y^{(n-1)}). \qquad (4.3)$$

Чтобы найти решение, дифференциальное уравнение (4.3) приводится к системе уравнений первого порядка. Для этого вводятся n новых неизвестных $y_1(t), y_2(t), ..., y_n(t)$. Причем

$$y_1(t) = y;$$

$$y_2(t) = y';$$

...

$$y_n(t) = y^{(n-1)}.$$

Это позволяет записать уравнение (4.3) в виде системы из n дифференциальных уравнений первого порядка:

$$y_1' = y_2;$$
$$y_2' = y_3;$$
$$\cdots$$
$$y_{n-1}' = y_n;$$
$$y_n' = f\left(t,\ y_1,\ y_2,\ \ldots,\ y_n\right),$$

решение которой можно найти процедурами библиотеки IMSL.

Например, обыкновенное дифференциальное уравнение (4.4) описывает движение перевернутого маятника

$$\theta'' = \frac{3}{2L}(g - A\varpi^2 \sin \varpi t)\sin\theta, \qquad (4.4)$$

где $\theta(t)$ - угол отклонения маятника в момент времени t; L - длина маятника; g - ускорение свободного падения; A и ϖ - соответственно амплитуда и частота вынужденных колебаний поддерживающего шарнира, описываемых уравнением $s = A\sin \varpi t$. Для преобразования уравнения (4.4) в систему из двух обыкновенных дифференциальных уравнений первого порядка введем неизвестные функции $y_1(t) = \theta(t)$ и $y_2(t) = \theta'(t)$. Тогда получим систему

$$y_1' = y_2;$$

$$y_2' = \frac{3}{2L}(g - A\varpi^2 \sin \varpi t)\sin y_1$$

с начальными условиями $y_1(t_0) = \theta(t_0)$ и $y_2(t_0) = \theta'(t_0)$.

Другой пример: дифференциальное уравнение Ван дер Поля $u'' + \mu(u^2 - 1)u' + u = 0$ - нелинейное обыкновенное дифференциальное уравнение второго порядка - сводится к дифференциальной алгебраической системе первого порядка

$$g_1 = y_2 - y_1' = 0;$$

$$g_2 = \left(1 - y_1^2\right)y_2 - \varepsilon\left(y_1 + y_2'\right) = 0,$$

где $\varepsilon = 1/\mu$, $y_1 = u$, решаемой подпрограммой DASPG.

4.1.5. УСТОЙЧИВОСТЬ РЕШЕНИЯ СИСТЕМЫ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

Напомним, что решение вычислительной задачи называется *устойчивым по входным данным*, если оно зависит от них непрерывным образом. Теоретически для устойчивой вычислительной задачи ее решение, если обеспечена достаточно высокая точность входных данных, можно найти со сколь угодно высокой точностью.

Оценка устойчивости системы дифференциальных уравнений может быть выполнена в результате анализа собственных значений *матрицы Якоби J* (*якобианом*) системы. Элемент матрицы $J_{ij} = \partial f_i / \partial y_j$. Так, в приведенной выше задаче о маятнике

$$J = \begin{pmatrix} 0 & 1 \\ 3(g - A\varpi^2 \sin \varpi t)\cos y_1 / (2L) & 0 \end{pmatrix}. \qquad (4.5)$$

Собственными значениями матрицы (4.5) являются корни квадратного уравнения

$$\lambda^2 - \frac{3}{2L}(g - A\varpi^2 \sin \varpi t)\cos y_1 = 0.$$

Можно показать, что устойчивостью управляет собственное значение с наибольшей вещественной частью [9] (в общем случае собственные значения - это комплексные числа). Причем положительные вещественные части обычно соответствуют областям с неустойчивыми (расходящимися) решениями. В то же время в большинстве практических задачах наличие у всех собственных значений отрицательных вещественных частей говорит об устойчивости системы.

В процессе вычислений якобиан меняется (зависит от t и решения y). Следовательно, меняются и его собственные значения. Причем в общем случае при различных t могут встречаться собственные значения как с положительными, так и с отрицательными вещественными частями.

4.1.6. СИСТЕМЫ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

Системы обыкновенных дифференциальных уравнений первого порядка, имеющие вид

$$y'(t) = \frac{dy}{dt} = f(t,\ y(t)), \tag{4.6}$$

где

$$y(t) = (y_1(t), ..., y_n(t))^T;\ f(t, y) = (f_1(t, y), ..., f_n(t, y))^T,$$

с начальными значениями $y(t_0)$ решаются процедурами IVPRK, IVMRK и IVPAG, которые находят $y(t)$ для $t > t_0$ или $t < t_0$. Процедуры IVPAG и DASPG также решают системы вида

$$Ay' = f(t, y),$$

где A - $n \times n$-матрица. Причем для IVPAG матрица не должна быть вырожденной.

В двухточечных краевых задачах с обыкновенными дифференциальными уравнениями дополнительные условия на решение налагаются в конечных точках исследуемого отрезка $[a, b]$. В подобного рода задачах используются процедуры BVPFD и BVPMS, решающие систему (4.6) с ограничениями

$$h_i(y_1(a), ..., y_n(a), y_1(b), ..., y_n(b)) = 0,\ i = 1, ..., n,$$

где f и $h = (h_1, ..., h_n)$ - пользовательские функции.

Замечание. При $n = 1$ названные процедуры библиотеки IMSL решают вместо системы одно обыкновенное дифференциальное уравнение.

4.1.7. ЖЕСТКИЕ ДИФФЕРЕНЦИАЛЬНЫЕ УРАВНЕНИЯ

Система (4.4) является *жесткой*, если некоторые собственные значения якобиана - матрицы $(\partial f_i / \partial y_j)$ - велики и имеют отрицательную вещественную часть. Такие задачи часто возникают при моделировании химических реакций, компоненты которых завершают взаимодействие в существенно различающиеся отрезки времени. Также жестким будет система, описывающая процесс разряда конденсаторов в цепи с широким диапазоном изменения постоянных времени отдельных ее участков. С особенностями решения жестких задач можно ознакомиться, например, в [1].

Отметим, однако, что определение жесткости задачи, базирующееся на оценке собственных значений якобиана, не является удовлетворительным. Поэтому на практике пользователи зачастую квалифицируют задачу как жесткую, если решатель дифференциальных уравнений, например IVPRK, оказывается неэффективным или не находит решения вовсе. Наиболее часто решатель признается неэффективным, если требуется большое число оценок функций f_i. В таких случаях следует употреблять процедуру IVPAG или DASPG.

4.1.8. ДИФФЕРЕНЦИАЛЬНЫЕ АЛГЕБРАИЧЕСКИЕ УРАВНЕНИЯ

Часто модель выражается неявной системой (или отдельным уравнением при $n = 1$) вида

$$g_i(t, y_1, \ldots, y_n, y'_1, \ldots, y'_n) = 0,\ i = 1, \ldots, n,$$

где g_i - пользовательская функция. Компактная запись системы такова:

$$g(t, y, y') = (g_1(t, y, y'), \ldots, g_n(t, y, y')) = 0. \qquad (4.7)$$

В качестве начальных задаются значения $y(t_0)$ и $y'(t_0)$. Система обыкновенных дифференциальных уравнений легко приводится к системе алгебраических дифференциальных уравнений путем задания

$$g(t, y, y') = f(t, y) - y'.$$

Для решения системы (4.7) индекса 1 или 0 применяется процедура DASPG. Метод решения системы (4.7) изложен в [30].

4.1.9. ДИФФЕРЕНЦИАЛЬНЫЕ УРАВНЕНИЯ В ЧАСТНЫХ ПРОИЗВОДНЫХ

Процедура MOLCH решает системы вида

$$\frac{\partial u_i}{\partial t} = f_i\left(x, t, u_1, \ldots, u_n, \frac{\partial u_1}{\partial x}, \ldots, \frac{\partial u_n}{\partial x}, \frac{\partial^2 u_1}{\partial x^2}, \ldots, \frac{\partial^2 u_n}{\partial x^2}\right)$$

с граничными условиями

$$\alpha_1^{(i)} u_i(a) + \beta_1^{(i)} \frac{\partial u_i}{\partial x}(a) = \gamma_1(t),$$

$$\alpha_2^{(i)} u_i(b) + \beta_{12}^{(i)} \frac{\partial u_i}{\partial x}(b) = \gamma_2(t)$$

и начальными значениями

$$u_i(x, t = t_0) = g_i(x)$$

для $i = 1, \ldots, n$. Здесь $f_i, g_i, \alpha_j^{(i)}, \beta_j^{(i)}$ - задаваемые пользователем функции и величины; $j = 1, 2$.

Процедуры FPS2H и FPS3H решают двумерные или трехмерные уравнения Лапласа (4.8), Пуассона (4.9) и Гельмгольца (4.10) (приводятся трехмерные уравнения):

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}=0; \tag{4.8}$$

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}=f(x,\ y,\ z); \tag{4.9}$$

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}=cu, \tag{4.10}$$

где $u = u(x, y, z)$; c - постоянное число.

Процедура FPS2H использует быстрый решатель уравнений в частных производных вида

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+cu=f(x,\ y).$$

Решение ищется на прямоугольной сетке с известными граничными условиями на каждой из четырех сторон. Константа c и функция f задаются пользователем. Процедура FPS3H решает аналогичное трехмерное уравнение.

Заметим, что уравнению Лапласа удовлетворяют так называемые *шаровые функции* - однородные многочлены от прямоугольных координат x, y и z с целой положительной степенью n. Их линейная комбинация представляет общий вид такого многочлена. Например,

$$u_2 = a(x^2 + z^2) + b(y^2 + z^2) + cxy + dyz + ezx,$$

где a, b, c, d, e - произвольные постоянные, - общий вид однородного многочлена степени 2.

4.1.10. ПЕРЕЧЕНЬ РЕШАЕМЫХ ПРОЦЕДУРАМИ IMSL ЗАДАЧ

В табл. 4.1 приведены виды дифференциальных уравнений, решаемых процедурами библиотеки IMSL. Употребляемые для их решения процедуры представлены в последней графе таблицы. Все процедуры, кроме FPS2H и FPS3H, могут использоваться для решения более одного дифференциального уравнения.

В последней строке таблицы указана задача, решаемая подпрограммой PDE_1D_MG, принадлежащей библиотеке IMSL 90 MP. Напомним, что математическая библиотека IMSL написана на языке программирования Фортран 77, в то время как подпрограммы библиотеки IMSL 90 MP реализованы на языке программирования Фортран 90.

Таблица 4.1. Дифференциальные уравнения и их системы, решаемые IMSL

Проблема	*Описание*	*Процедуры*
$Ay' = f(t, y)$, $y(t_0) = y_0$	A - это либо матрица общего вида, либо симметрическая положительно определенная матрица, либо ленточная или симметрическая положительно определенная ленточная матрица	IVPAG
То же	Жесткая или требующая большого объема вычислений $f(t, y)$ задача. Применяется якобиан, либо явно заданный, либо аппроксимируемый разделенными разностями	IVPAG
$y' = f(t, y)$, $y(t_0) = y_0$	Нежесткая задача, требующая высокоточного решения. Используется конечно-разностный метод Адамса решения задачи Коши для систем дифференциальных уравнений первого порядка	IVPAG
То же	Нежесткая задача, требующая решения умеренной точности	IVPRK, IVMRK
$y' = f(t, y)$, $h(y(a), y(b)) = 0$	Двухточечная краевая задача. Решается методом конечных разностей	BVPFD
То же	Двухточечная краевая задача. Решается методом пристрелки	BVPMS
$g(t, y, y') = 0$ при заданных $y(t_0)$ и $y'(t_0)$	Жесткая система дифференциальных алгебраических уравнений индекса 1 или 0	DASPG
$u_t = f(x, t, u, u_x, u_{xx})$, $\alpha_1 u(a) + \beta_1 u_x(a) = \gamma_1(t)$, $\alpha_2 u(b) + \beta_2 u_x(b) = \gamma_2(t)$	Используется метод прямых, с применением кубических сплайнов и обыкновенных дифференциальных уравнений	MOLCH
$u_{xx} + u_{yy} + cu = f(x, y)$ на прямоугольной сетке с заданными u или u_n на каждой ее границе	Используется быстрый решатель уравнений (4.8) - (4.10), основанный на конечно-разностной схеме на однородной сетке	FPS2H
$u_{xx} + u_{yy} + u_{zz} + cu = f(x, y, z)$ в прямоугольном параллелепипеде с заданными u или u_n на каждой его грани	То же	FPS3H

$-(pu')' + qu = \lambda ru$ с граничными условиями вида $\alpha_1 u(a) - \alpha_2(pu'(a)) =$ $\lambda(\alpha'_1 u(a) - \alpha'_2(pu'(a)))$, $\beta_1 u(b) + \beta_2(pu'(b)) = 0$	Задача Штурма - Лиувилля. К решению этого уравнения сводятся некоторые задачи математической физики	SLEIG
$u_t = \frac{\partial u}{\partial t} = f(u, x, t)$, $x_L < x < x_R, t > t_0$	Системы дифференциальных уравнений в частных производных	PDE_1D_MG

4.2. ЗАДАЧА КОШИ

4.2.1. ПОДПРОГРАММА IVPRK (DIVPRK)

Решает систему дифференциальных уравнений с начальными условиями (задачу Коши), используя метод Рунге - Кутты - Вернера пятого или шестого порядка точности. Имеет вызов

CALL IVPRK(*ido, n, fcn, t, tend, tol, param, y*)

Параметры подпрограммы IVPRK:

Пользовательская подпрограмма: fcn.

Входные: n, tend, tol.

Входные/выходные: ido, t, param, y.

ido - флаг, характеризующий стадию вычислений. Принимает следующие значения:

- *ido* = 1 - начальный вход (первый вызов);
- *ido* = 2 - нормальный повторный вызов;
- *ido* = 3 - последний вызов, приводящий к освобождению рабочей области;
- *ido* = 4 - возврат из-за прерывания 1;
- *ido* = 5 - возврат из-за прерывания 2; шаг принимается;
- *ido* = 6 - возврат из-за прерывания 2; шаг не принимается.

Обычно первый вызов выполняется с *ido* = 1. При этом автоматически выделяется память - рабочая область. Затем подпрограмма устанавливает *ido* = 2, и это значение используется для всех повторных вызовов, кроме последнего, который выполняется с *ido* = 3. Последний вызов с *ido* = 3 освобождает рабочую область. Интегрирование на последнем шаге не выполняется. Комментарий отноосительно других значений *ido* см. ниже.

n - число дифференциальных уравнений.

fcn - пользовательская подпрограмма, выполняющая оценку функции. Должна быть снабжена атрибутом EXTERNAL. Имеет вызов

CALL *fcn*(*n, t, y, yprime*)

Параметры подпрограммы fcn:

n - число дифференциальных уравнений.

t - независимая переменная.

y - массив размера *n*, содержащий значения зависимых переменных (функций).

yprime - массив размера *n*, содержащий значения *y'*, вычисленные для (*t, y*).

Параметры *n, t* и *y* являются *входными*, параметр *yprime* - *выходной*.

Продолжение описания параметров подпрограммы IVPRK:

t - независимая переменная. На входе содержит начальное значение. На выходе, если не произошло ошибки, замещается на *tend*.

tend - значение *t*, в котором надо получить решение. Причем *tend* может быть меньше начального значения *t*.

tol - допуск для контроля ошибок. Выполняется попытка контролировать норму локальной ошибки таким образом, что глобальная ошибка оказывается сравнимой с *tol*.

param - вещественный массив размера 50, содержащий необязательные параметры, управляющие процессом вычисления. Если элемент массива *param* равен нулю, то для соответствующего параметра IVPRK использует установленное по умолчанию значение. Параметры, определяющие размер шага, применяются в направлении интегрирования. Могут быть заданы следующие параметры:

- *param*(1) = HINIT - начальный размер шага; по умолчанию равен 10.0 * MAX(AMACH(1), AMACH(4) * MAX(ABS(*tend*), ABS(*t*)));
- *param*(2) = HMIN - минимальный размер шага; по умолчанию равен 0.0;
- *param*(3) = HMAX - максимальный размер шага; по умолчанию равен 2.0;
- *param*(4) = MXSTEP - максимально допустимое число шагов; по умолчанию равно 500;
- *param*(5) = MXFCN - максимально допустимое число оценок функции; по умолчанию ограничений на число оценок нет;
- *param*(6) - не используется;

- *param*(7) = INTRP1 - если отличен от нуля, то подпрограмма IVPRK возвращает *ido* = 4 (см. ниже комментарий 2). По умолчанию INTRP1 = 0;
- *param*(8) = INTRP2 - если отличен от нуля, то подпрограмма IVPRK возвращает *ido* = 5 после каждого успешного шага и *ido* = 6 после каждого неуспешного шага (см. ниже комментарий 2). По умолчанию INTRP2 = 0;
- *param*(9) = SCALE - мера масштабирования среднего значения нормы матрицы Якоби. По умолчанию SCALE = 1.0;
- *param*(10) = INORM - переключатель, задающий норму для вычисления ошибки e_i - абсолютной величины оценки ошибки определения $y_i(t)$.

  Формула для вычисления нормы задается значением INORM.

  Если INORM = 0, то применяется формула MIN(*абсолютная ошибка*, *относительная ошибка*) = MAX(e_i / w_i), $i = 1, ..., n$, где w_i = MAX($|y_i(t)|$, 1.0);

  INORM = 1, то вычисляется абсолютная ошибка, равная MAX(e_i), $i = 1, ..., n$;

  INORM = 2, то используется формула - MAX(e_i / w_i), $i = 1, ..., n$, где w_i = MAX($|y_i(t)|$, FLOOR), FLOOR - это *param*(11);

  INORM = 3, то употребляется масштабированная евклидова норма, определяемая как $ymax = \sqrt{\sum_{i=1}^{n} e_i^2 / w_i^2}$, где w_i = MAX($|y_i(t)|$, 1.0).

  Значение по умолчанию INORM = 0;
- *param*(11) = FLOOR - используется при вычислении нормы, когда INORM = 2. Значение по умолчанию - 1.0;
- *param*(12) - *param*(30) - не используются.

Следующие элементы массива *param* определяются программой:

- *param*(31) = HTRIAL - текущий размер шага;
- *param*(32) = HMINC - вычисленное минимально допустимое значение шага;
- *param*(33) = HMAXC - вычисленное максимально допустимое значение шага;
- *param*(34) = NSTEP - число выполненных шагов;
- *param*(35) = NFCN - число вызовов функции;
- *param*(36) - *param*(50) - не используются.

y - вектор размера *n* зависимых переменных (значений функций). На входе *y* содержит начальные значения, на выходе - приблизительное решение.

Автоматически для решения предоставляется память:

- $10n$ байт в случае IVPRK;
- $20n$ байт в случае DIVPRK.

Память можно выделить явно, применив I2PRK (DI2PRK):

CALL I2PRK(*ido, n, fcn, t, tend, tol, param, y, vnorm, wk*)

Дополнительные параметры подпрограммы I2PRK:

vnorm - подпрограмма, вычисляющая норму ошибки. Подпрограмма может быть предоставлена пользователем или IMSL. В последнем случае она имеет имя I3PRK (DI3PRK). В обоих случаях подпрограмма должна иметь атрибут EXTERNAL.

wk - вещественный рабочий массив размера $10n$. Содержимое *wk* не должно изменяться после первого вызова с *ido* = 1 до последнего с *ido* = 3.

Подпрограмма *vnorm* имеет вызов

CALL *vnorm*(*n, v, y, ymax, enorm*)

Параметры подпрограммы vnorm:

Описание параметров *n* и *y* см. выше.

v - вектор размера *n*, норма которого вычисляется.

ymax - вектор размера *n*, содержащий максимальные значения $|y(t)|$.

enorm - норма вектора *v*.

Параметры *n, v, y, ymax* являются *входными*, параметр *enorm* - *выходной*.

Комментарии:

1. Возможные информационные ошибки:

Тип	*Код*	*Описание*
4	1	Нельзя удовлетворить условиям, заданным для величины ошибки. Параметр *tol* может быть слишком маленьким
4	2	Использовано максимально допустимое число оценок функции
4	3	Выполнено максимально допустимое число шагов. Задача может быть жесткой

2. Если *param*(7) отличен от нуля, подпрограмма завершится с *ido* = 4 и продолжит вычисления с точки прерывания, при последующем вызове с *ido* = 4. Если *param*(8) отличен от нуля, подпрограмма завершит вычисления немедленно после того, как решит, следует ли принимать результат последнего шага или нет. Если подпрограмма планирует при-

нять результат, то она использует *ido* = 5, или *ido* = 6 - в противном случае. Пользователь может заменить значение *ido* с числа 6 на 5, чтобы заставить программу принять результат. После прерывания пользователь может проанализировать параметры *ido*, *htrial*, *nstep*, *nfcn*, *t* и *y*. Массив *y* содержит вычисленные на последнем шаге значения *y*(*t*), принятые или нет.

Описание:

Подпрограмма IVPRK находит приблизительное решение системы дифференциальных уравнений первого порядка вида $y' = f(t, y)$ с заданными начальными данными; старается поддерживать величину глобальной ошибки в пределах заданного допуска; эффективна для нежестких систем, в которых оценка производных не требует больших вычислительных затрат.

Подпрограмма IVPRK основана на коде, приведенном в [35], использует формулы Рунге - Кутты пятого или шестого порядка точности, построенные Вернером.

Пример 1. Рассматривается задача "хищник - жертва" с зайцами и лисицами. Пусть *r* - плотность зайцев, а *f* - лисиц. Предположим, что при отсутствии взаимодействия "хищник - жертва" популяция зайцев возрастает пропорционально их числу, а лисицы будут умирать от голода также со скоростью, пропорциональной их количеству. Математически это выглядит так:

$$r' = 2r;$$
$$f' = -f.$$

Пусть теперь скорость, с которой лисицы поедают зайцев, равна $2rf$, а скорость возрастания популяции лисиц в результате уничтожения ими зайцев равна rf. Поэтому модель "хищник - жертва" такова:

$$r' = 2r - 2rf;$$
$$f' = -f + rf.$$

Зададим следующие начальные условия: $r(0) = 1$ и $f(0) = 3$ на отрезке $0 \le t \le 10$. В программе $y(1) = r$ и $y(2) = f$. Параметр *param* первоначально обнуляется. Затем в результате задания *param*(10) = 1.0 устанавливается режим контролирования абсолютной ошибки. Последний вызов IVPRK с *ido* = 3 освобождает выделенную при первом вызове IVPRK рабочую область. Правда, в этом примере освобождать рабочую область необязательно, так как после нахождения решения программа завершается. Рабочая область подлежит освобождению, если в дальнейшем программа обращается к процедурам IMSL.

```
program ivprk1
 use dfimsl
 integer(4), parameter :: mxparm = 50, n = 2
```

```
 integer(4) :: ido, istep, nout
 real(4) :: param(mxparm), t, tend, tol, y(n)
 external fcn
 t = 0.0                                  ! Начальные условия
 y(1) = 1.0
 y(2) = 3.0
 tol = 0.0005                             ! Допуск для оценки ошибки
 param = 0.0                              ! Действуем по умолчанию
 param(10) = 1.0                          ! Контролируем абсолютную ошибку
 ! Вывод заголовка таблицы результатов
 print "(4x, 'istep', 5x, 'time', 9x, 'y1', 11x, 'y2')"
 ido = 1
 istep = 0
 do while(istep < 10)
  istep = istep + 1
  tend = istep
  call ivprk(ido, n, fcn, t, tend, tol, param, y)
  print '(i6, 3f12.3)', istep, t, y
  if(istep == 10) ido = 3                 ! Освобождаем память
 end do
end program ivprk1

subroutine fcn(n, t, y, yprime)
 integer(4) :: n
 real(4) :: t, y(n), yprime(n)
 yprime(1) = 2.0 * y(1) - 2.0 * y(1) * y(2)
 yprime(2) = -y(2) + y(1) * y(2)
end subroutine fcn
```

Результат:

istep	time	y1	y2
1	1.000	0.078	1.465
2	2.000	0.085	0.578
3	3.000	0.292	0.250
4	4.000	1.449	0.187
5	5.000	4.046	1.444
6	6.000	0.176	2.256
7	7.000	0.066	0.908
8	8.000	0.148	0.367
9	9.000	0.655	0.188
10	10.000	3.157	0.352

Пример 2. Решается умеренно жесткая система из двух дифференциальных уравнений из тестовой серии [24]. Пример демонстрирует неэффективность нежесткого решателя, проявляющуюся в большом числе вызовов функции *fcn*. Та же точность достигается на жестком решателе, использующем дифференцирование назад, за существенно меньшее число вызовов (см. пример 2 для IVPAG).

$$y'_1 = -y_1 - y_1y_2 + k_1y_2, \qquad y_1(0) = 1.0;$$

$$y'_2 = -k_2y_2 + k_3(1 - y_2)\,y_1, \qquad y_2(0) = 0.0;$$

$k_1 = 294.0$, $k_2 = 3.0$, $k_3 = 0.01020408$, $tend = 240.0$.

```
program ivprk2
 use dfimsl
 integer(4), parameter :: mxparm = 50, n = 2
 integer(4) :: ido, istep
 real(4) :: param(mxparm), t, tend, tol, y(n)
 external fcn
 t = 0.0                              ! Начальные условия
 y(1) = 1.0
 y(2) = 0.0
 tol = 0.001                          ! Допуск для оценки ошибки
 param = 0.0                          ! Действуем по умолчанию
 param(10) = 1.0                      ! Контролируем абсолютную ошибку
 ! Вывод заголовка таблицы результатов
 print "(4x, 'istep', 5x, 'time', 9x, 'y1', 11x, 'y2')"
 ido = 1
 istep = 0
 do while(istep < 240)
  istep = istep + 24
  tend = istep
  call ivprk(ido, n, fcn, t, tend, tol, param, y)
  print '(i6, 3f12.3)', istep / 24, t, y
  if(istep == 240) ido = 3            ! Освобождаем память
 end do
 ! Число вызовов подпрограммы fcn
 print "(1x, 'Number of fcn calls with IVPRK = ', f6.0)", param(35)
end program ivprk2

subroutine fcn(n, t, y, yprime)
 integer(4) :: n
 real(4) :: t, y(n), yprime(n)
 real(4) :: ak1 = 294.0, ak2 = 3.0, ak3 = 0.01020408
```

```
 yprime(1) = -y(1) - y(1) * y(2) + ak1 * y(2)
 yprime(2) = -ak2 * y(2) + ak3 * (1.0 - y(2)) * y(1)
end subroutine fcn
```

Результат:

istep	time	y1	y2
1	24.000	0.688	0.002
2	48.000	0.634	0.002
3	72.000	0.589	0.002
4	96.000	0.549	0.002
5	120.000	0.514	0.002
6	144.000	0.484	0.002
7	168.000	0.457	0.002
8	192.000	0.433	0.001
9	216.000	0.411	0.001
10	240.000	0.391	0.001

Number of fcn calls with IVPRK = 2125.

4.2.2. ПОДПРОГРАММА IVMRK (DIVMRK)

Решает задачу Коши $y' = f(t, y)$ для обыкновенных дифференциальных уравнений с использованием методов Рунге - Кутты различных порядков точности. Имеет вызов

CALL IVMRK(*ido, n, fcn, t, tend, y, yprime*)

Параметры подпрограммы IVMRK:

Пользовательская подпрограмма: fcn.

Входные: n, tend.

Входные/выходные: ido, t, y, yprime.

ido - флаг, характеризующий стадию вычислений. Параметр, когда он равен 1, 2 или 3, задает те же действия, что и при вызовах подпрограммы IVPRK, рассмотренной в предшествующем разделе. Принимает также и иные значения:

- *ido* = 4 - возврат после выполнения шага;
- *ido* = 5 - возврат для выполнения оценки производной (дифференцирование назад).

Обычно первый вызов выполняется с *ido* = 1. При этом автоматически выделяется память - рабочая область. Затем подпрограмма устанавливает

ido = 2, и это значение используется для всех повторных вызовов, кроме последнего, который выполняется с *ido* = 3. Последний вызов с *ido* = 3 освобождает рабочую область. Интегрирование на последнем шаге не выполняется. Комментарий относительно других значений *ido* см. ниже.

n - число дифференциальных уравнений.

fcn - пользовательская подпрограмма, выполняющая оценку функции. Должна быть снабжена атрибутом EXTERNAL. Описание *fcn* и ее параметров см. в предшествующем разделе.

t - независимая переменная. На входе содержит начальное значение. На выходе, если не произошло ошибки, замещается на *tend*.

tend - значение *t*, в котором надо получить решение. Причем *tend* может быть меньше начального значения *t*.

y - вектор размера *n* зависимых переменных (значений функций). На входе *y* содержит начальные значения, на выходе - приблизительное решение.

yprime - массив размера *n*, содержащий значения *y'*, вычисленные для (*t*, *y*).

Автоматически для решения предоставляется память:

- 42*n* + 50 байт в случае IVMRK;
- 84*n* + 100 байт в случае DIVMRK.

Память можно выделить явно, применив I2MRK (DI2MRK):

```
CALL I2MRK(ido, n, fcn, t, tend, y, yprime, tol, thres, param,          &
     ymax, rmserr, work, Lwork)
```

Дополнительные параметры подпрограммы I2MRK:

Входные: thres, Lwork.

Входные/выходные: tol, param.

Выходные: ymax, rmserr.

Рабочий массив: work.

tol - допуск для контроля ошибок.

thres - вектор размера *n*, где *thres*(*i*) - пороговое значение для компонента решения *y*(*i*). Выбирается таким образом, что значение *y*(*i*) не существенно, когда *y*(*i*) < *thres*(*i*). Причем *thres*(*i*) ≥ SQRT(AMACH(4)).

param - вещественный массив размера 50, содержащий необязательные параметры. Если элемент *param* равен нулю, то для соответствующего параметра IVMRK использует установленное по умолчанию значение. Могут быть заданы следующие параметры:

- *param*(1) = HINIT - начальный размер шага. Должен быть выбран на отрезке 10.0 * AMACH(4) ≤ HINIT ≤. 0.01. По умолчанию задан режим автоматического выбора величины шага;
- *param*(2) = METHOD - задает используемые в методе Рунге - Кутты пары. При значениях 1, 2 и 3 будут использоваться соответственно пары (2, 3), (4, 5) и (7, 8).

  По умолчанию

  METHOD = 1, если $10^{-2} \geq tol > 10^{-4}$;

  METHOD = 2, если $10^{-4} \geq tol > 10^{-6}$;

  METHOD = 3, если $10^{-6} \geq tol$
- *param*(3) = ERREST. Если ERREST = 1, то выполняется попытка добиться минимальной ошибки - разницы между численным и истинным решениями. Цена такого режима (по сравнению с заданием ERREST = 0) - приблизительно двойное увеличение времени счета при METHOD = 2 или METHOD = 3 и тройное при METHOD = 1. По умолчанию ERREST = 0;
- *param*(4) = INTRP - обеспечивает, если не равен нулю, возврат с *ido* = 4. См. также приводимый ниже комментарий 2. По умолчанию INTRP = 0;
- *param*(5) = RCSTAT. Если RCSTAT ≠ 0, то для вычисления производной используется дифференцирование назад. См. также приводимый ниже комментарий 2. По умолчанию RCSTAT = 0;
- *param*(6) - *param*(30) - не используются.

Следующие элементы массива *param* возвращаются программой:

- *param*(31) = HTRIAL - текущий размер шага;
- *param*(32) = NSTEP - число выполненных шагов;
- *param*(33) = NFCN - число вызовов функции;
- *param*(34) = ERRMAX - максимальная взвешенная ошибка по всем компонентам решения и всем выполненным шагам, начиная с момента времени *t* до текущего момента интегрирования;
- *param*(35) = TERRMX - первое значение независимой переменной, в которой ошибка приобретает максимальное значение ERRMAX;
- *param*(36) - *param*(50) - не используются.

ymax - вектор размера *n*, в котором *ymax*(*i*) - максимальное (на текущий момент времени) значение ABS(*y*(*i*)).

rmserr - вектор размера *n*, в котором *rmserr*(*i*) содержит приблизительное значение средней ошибки для решения *i*, *i* = 1, ..., *n*. Усреднение выполняется от точки *t* до текущей точки интегрирования. Значение параметра *rmserr* определяется, если *param*(3) = 1.

work - вещественный рабочий массив размера не менее 39*n*. Содержимое *work* не должно изменяться после первого вызова с *ido* = 1 до последнего с *ido* = 3.

Lwork - размер рабочего массива *work*.

Комментарии:

1. Возможные информационные ошибки:

Тип	*Код*	*Описание*
4	1	Нельзя удовлетворить погрешности, заданной параметрами *tol* и *thres*, используя текущие точность и METHOD. Задание большего значения METHOD разрешит работу с большей погрешностью с текущей точностью. Интегрирование должно быть выполнено заново
4	2	Значение глобальной ошибки может быть неверным после текущей точки интегрирования *t*. Причиной этого может быть либо слишком высокая или низкая заданная точность, либо недостаточная гладкость *f*(*t*, *y*) при значении *t* непосредственно после *tend*. Такая ошибка означает, что нельзя интегрировать после *tend*, если работа выполняется с *param*(3) = 1

2. Если *param*(4) отличен от нуля, подпрограмма завершится с *ido* = 4 и возобновит вычисления от точки прерывания, если вызвать ее с *ido* = 4. Параметры, которые обычно анализируются при таком вызове, - это *ido*, HTRIAL, NSTEP, NFCN, *t* и *y*. Массив *y* содержит значения *y*(*t*) для последнего шага, принятые или нет.

3. Если *param*(5) отличен от нуля, подпрограмма завершится с *ido* = 5. Далее следует вычислить производные в точке *t*, разместить результат в *yprime* и вызвать IVMRK вновь. Вместо *fcn* можно употребить функцию I40RK (DI40RK).

Описание:

Подпрограмма IVMRK находит приблизительное решение системы дифференциальных уравнений первого порядка вида $y' = f(t, y)$ с заданными начальными данными. При решении контролируется в соответствии с заданным допуском относительная локальная ошибка. Для повышения эффективности доступны схемы Рунге - Кутты порядков 3, 5 и 8. В качестве необязательного параметра в IVMRK можно передать значения вектора y', полученные в ре-

зультате дифференцирования назад, не прибегая к применению пользовательской подпрограммы *fcn*. Дифференцирование назад особенно полезно в приложениях, в которых вычисление $f(t, y)$ трудоемко. Также можно задать режим оценки глобальной ошибки интегрирования.

Подпрограмма IVMRK основана на коде, предложенном в [16].

Пример 1. Решается небольшая система из тестового набора, имеющегося в [24].

$$y'_1 = -y_1 + y_2, \qquad y_1(0) = 2.0;$$

$$y'_2 = y_1 - 2y_2 + y_3, \qquad y_2(0) = 0.0;$$

$$y'_3 = y_2 - y_3, \qquad y_3(0) = 1.0.$$

```
program ivmrk1
 use dfimsl
 integer(4), parameter :: n = 3
 integer(4) :: ido
 real(4) :: t, tend, y(n), yprime(n)
 external fcn
 t = 0.0; tend = 20.0                    ! Начальные значения
 y(1) = 2.0; y(2) = 0.0; y(3) = 1.0
 ido = 1
 call ivmrk(ido, n, fcn, t, tend, y, yprime)
 ido = 3                                 ! Освобождаем рабочую область
 call ivmrk(ido, n, fcn, t, tend, y, yprime)
 call wrrrn('y', n, 1, y, n, 0)          ! Вывод результата
end program ivmrk1

subroutine fcn(n, t, y, yprime)          ! Подпрограмма, выполняющая оценку функций
 integer(4) :: n                         ! (правых частей системы)
 real(4) :: t, y(*), yprime(*)
 yprime(1) = -y(1) + y(2)
 yprime(2) = y(1) - 2.0 * y(2) + y(3)
 yprime(3) = y(2) - y(3)
end subroutine fcn
```

Результат:

```
          y
1     1.000
2     1.000
3     1.000
```

Пример 2. Решается та же, что и в примере 2 из предыдущего раздела, умеренно жесткая задача. Подпрограмма IVMRK не является жестким решателем, но из примера видно, что она более эффективна, чем IVPRK (в смысле количества оценок производной). Также в этом примере используется дифференцирование назад.

```
program ivmrk2
 use dfimsl
 integer(4), parameter :: mxparm = 50, n = 2
 integer(4) :: ido, istep, lwork
 real(4) :: param(mxparm), rmserr(n), t, tend, thres(n), tol, work(1000), y(n), ymax(n),
yprime(n)
 real(4) :: ak1 = 294.0, ak2 = 3.0, ak3 = 0.01020408
 t = 0.0                                  ! Начальные условия
 y(1) = 1.0; y(2) = 0.0
 tol = 0.001                              ! Допуск для оценки ошибки
 thres = amach(4)                         ! Пороговые значения
 param = 0.0                              ! Действуем по умолчанию
 Lwork = 1000
 ! Выполняем оценку производной путем дифференцирования назад
 param(5) = 1
 ido = 1
 ! Вывод заголовка таблицы результатов
 write(*,"(3x, 'istep', 5x, 'time', 9x, 'y1', 10x, 'y2')")
 istep = 24
 do while(istep <= 240)
  tend = istep
  call i2mrk(ido, n, i40rk, t, tend, y, yprime, tol, thres,                    &
       param, ymax, rmserr, work, Lwork)
  if(ido == 5) then
   ! Оценка производных
   yprime(1) = -y(1) - y(1) * y(2) + ak1 * y(2)
   yprime(2) = -ak2 * y(2) + ak3 * (1.0 - y(2)) * y(1)
  else if(istep <= 240) then
   ! Интегрируем в 10 равномерно расположенных точках
   write(*, '(i6, 3f12.3)') istep / 24, t, y
   if(istep == 240) ido = 3               ! Освобождаем память
   istep = istep + 24
  end if
 end do
 ! Выводим число оценок производной
 write(*,"(/, 4x, 'Number of derivative evaluations with IVMRK =', f6.0)") param(33)
end program ivmrk2
```

Результат:

istep	time	y1	y2
1	24.000	0.688	0.002
2	48.000	0.634	0.002
3	72.000	0.589	0.002
4	96.000	0.549	0.002
5	120.000	0.514	0.002
6	144.000	0.484	0.002
7	168.000	0.457	0.002
8	192.000	0.433	0.001
9	216.000	0.411	0.001
10	240.000	0.391	0.001

Number of derivative evaluations with IVMRK = 1387

Пример 3. Пример показывает, как следует обрабатывать исключения. Функции имеют точки разрыва. Пример взят из [24]. Оценка глобальной ошибки не может быть выполнена, поскольку функции имеют разрывы. При обнаружении этого факта подпрограмма отключает режим вычисления оценки глобальной ошибки и интегрирование осуществляется заново. Задаваемый первоначально допуск для относительной ошибки оказывается слишком малым. Если же интегрирование невозможно из-за того, что нельзя обеспечить заданную точность, то допуск увеличивается и интегрирование выполняется заново, причем также осуществляется оценка глобальной ошибки. Включение режима оценки глобальной ошибки объясняется тем, что прежний сбой при ее вычислении может произойти не по причине чрезмерно высокой затребованной точности, а недостаточной гладкости производных. Когда же интегрирование завершается успешно, то выводятся окончательные относительная ошибка, допуск и сообщение о возможности или невозможности выполнения оценки глобальной ошибки.

$$y_1' = y_2;$$

$$y_2' = \begin{cases} 2ay_2 - \left(\pi^2 + a^2\right) y_1 + 1, & [\ t\ ] - \text{четное число;} \\ 2ay_2 - \left(\pi^2 + a^2\right) y_1 - 1, & [\ t\ ] - \text{нечетное число;} \end{cases}$$

$$y_1(0) = 0, \quad y_2(0) = 0, a = 0.1,$$

где $[x]$ - целая часть числа x.

```
program ivmrk3
 use dfimsl
 integer(4), parameter :: n = 2
 integer(4) :: ido, lwork, nout
 real(4) :: param(50), rmserr(n), t, tend, thres(n), tol, work(100), y(n), ymax(n), yprime(n)
 logical(4) :: GoOn = .true.
 external fcn
 ! Будем продолжать вычисления и при возникновении фатальных ошибок
 call erset(4, -1, 0)
 lwork = 100                              ! Параметры для I2MRK
 tol = sqrt(amach(4))
 param = 0.0
 thres = sqrt(amach(4))
 tend = 20.0
 ! Включаем режим оценки глобальной ошибки
 param(3) = 1
 do while(GoOn)
  t = 0.0e0                               ! Начальные значения
  y(1) = 0.0; y(2) = 0.0
  ido = 1
  call i2mrk(ido, n, fcn, t, tend, y, yprime, tol, thres, param, ymax, rmserr, work, lwork)
  GoOn = .false.
  if(iercd( ) == 32) then
   ! Нельзя обеспечить запрошенную точность, поэтому увеличиваем допуск,
   ! активизируем режим оценки глобальной ошибки и повторяем интегрирование
   tol = 10.0 * tol
   param(3) = 1
   write(*, 99995) tol
   GoOn = .true.                          ! Повторяем интегрирование
  else if(iercd( ) == 34) then
   ! Не удается оценить глобальную ошибку
   ! Нельзя далее продолжать интегрирование, поэтому отключаем режим
   ! оценки глобальной ошибки и повторяем вычисления
   write (*, 99996)
   param(3) = 0
   GoOn = .true.                          ! Повторяем интегрирование
  end if
 end do
 ! Последний вызов. Освобождаем рабочую область памяти
 ido = 3
 call i2mrk(ido, n, fcn, t, tend, y, yprime, tol, thres,                    &
       param, ymax, rmserr, work, lwork)
 ! Вывод результата
```

```
 write(*, 99997) tol
 if(param(3) == 1) then
   write (*, 99998)
 else
   write(*, 99999)
 end if
 call wrrrn('y', n, 1, y, n, 0)
 99995 format(/, 'Changing tolerance to ', e9.3, ' and restarting ...', &
       /, 'Also (re)enabling global error assessment', /)
 99996 format(/, 'Disabling global error assessment and restarting ...', /)
 99997 format(/, 72('-'), //, 'Solution obtained with tolerance = ', e9.3)
 99998 format('Global error assessment is available')
 99999 format('Global error assessment is not available')
end program ivmrk3

subroutine fcn(n, t, y, yprime)
 use dfimsl
 integer(4) :: n
 real(4) :: t, y(*), yprime(*)
 real(4) :: a, pi
 logical(4) :: first
 save first, pi
 data first / .true. /
 if(first) then
   pi = const('pi')
   first = .false.
 end if
 a = 0.1
 yprime(1) = y(2)
 if(mod(int(t), 2) == 0) then
   yprime(2) = 2.0 * a * y(2) - (pi * pi + a * a) * y(1) + 1.0
 else
  yprime(2) = 2.0 * a * y(2) - (pi * pi + a * a) * y(1) - 1.0
 end if
end subroutine fcn
```

Результат:

```
*** FATAL ERROR 34 from I2MRK. The global error assessment may not
*** be reliable for T past 9.994789E-01. The integration is being terminated.
Disabling global error assessment and restarting ...
*** FATAL ERROR 32 from I2MRK. In order to satisfy the error
*** requirement I6MRK would have to use a step size of 3.322625E-06
*** at TNOW = 9.999973E-01. This is too small for the current precision.
```

```
Changing tolerance to 0.345E-02 and restarting ...
Also (re)enabling global error assessment.
*** FATAL ERROR 34 from I2MRK. The global error assessment may
*** not be reliable for T past 9.985998E-01. The integration is being terminated.
Disabling global error assessment and restarting ...
--------------------------------------------------------------------------
Solution obtained with tolerance = 0.345E-02
Global error assessment is not available

          y
1     -12.28
2       0.99
```

4.2.3. ПОДПРОГРАММА IVPAG (DIVPAG)

Решает задачу Коши $y' = f(t, y)$ для обыкновенных дифференциальных уравнений, используя либо метод Адамса - Мултона, либо метод Гира дифференцирования назад с автоматическим выбором шага интегрирования и порядка метода. Имеет вызов

CALL IVPAG(*ido, n, fcn, fcnj, a, t, tend, tol, param, y*)

Параметры подпрограммы IVPAG:

Пользовательские подпрограммы: fcn, fcnj.

Входные: n, a, tend, tol.

Входные/выходные: ido, t, param, y.

ido - флаг, характеризующий стадию вычислений. Параметр, когда он равен 1, 2, 3, 4, 5 или 6, задает те же действия, что и при вызовах подпрограммы IVPRK, рассмотренной в предшествующем разделе. Значение *ido* = 7 говорит о том, что выполнен возврат для получения новой матрицы *A*. После изменения матрицы *A* подпрограмма IVPAG должна быть вызвана вновь. При этом не должны изменяться другие параметры IVPAG, включая *ido*. Заметим, что значение *ido* = 7 возвращается IVPAG, если PARAM(19) = 2.

n - число дифференциальных уравнений.

fcn - пользовательская подпрограмма, выполняющая оценку функции. Должна иметь атрибут EXTERNAL. Описание *fcn* дано при разборе подпрограммы IVPRK.

fcnj - пользовательская подпрограмма, осуществляющая вычисление якобиана. Имеет вызов

CALL *fcnj*(*n, t, y, dypdy*)

Параметры подпрограммы fcnj:

n - число дифференциальных уравнений.

t - независимая переменная.

y - массив размера *n*, содержащий значения зависимой переменной *y*(*t*).

dypdy - массив, вид которого определяется PARAM(14). Содержит требуемые частные производные $\partial f_i / \partial y_i$.

Эти производные должны вычисляться при текущих значениях (*t*, *y*). Когда якобиан плотно заполнен (MTYPE = 0 или MTYPE = 2), ведущее измерение *dypdy* равно *n*. Когда же якобиан можно представить в виде ленточной матрицы, то MTYPE задается равным единице, а ведущее измерение массива *dypdy* - равным 2*nlc* + *nuc* + 1. Если же матрица является симметрической ленточной и положительно определенной, то MTYPE задается равным трем, а ведущее измерение *dypdy* - равным *nuc* + 1.

Параметры *n, t* и *y* являются *входными*, параметр *dypdy* - *выходной*.

Подпрограмма *fcnj* в вызывающей программной единице должна иметь атрибут EXTERNAL. Если PARAM(19) отличен от нуля, то *fcnj* должна вычислять якобиан правой стороны уравнения $Ay' = f(t, y)$. Заметим, что подпрограмма *fcnj* употребляется, если PARAM(13) = 1.

Продолжение описания параметров подпрограммы IVPAG:

a - массив, представляющий матрицу *A*, используемую при решении неявных систем. Матрица *a* адресуется, только когда PARAM(19) отличен от нуля. Тип данных матрицы определяется PARAM(14). Матрица *a* не должна быть вырожденной, и значение MITER должно быть равно 1 или 2 (см. ниже комментарий 3).

t - независимая переменная. На входе содержит начальное значение. На выходе, если не произошло ошибки, замещается на *tend*.

tend - значение *t*, в котором надо получить решение. Причем *tend* может быть меньше начального значения *t*.

tol - допуск для контроля ошибок. Выполняется попытка контролировать норму локальной ошибки таким образом, что глобальная ошибка не превышает *tol*.

param - вещественный массив, размер которого равен 50. Содержит необязательные параметры, управляющие ходом решения. Ели параметр равен нулю, то используется заданное по умолчанию значение. Параметры, определяющие размер шага, применяются в направлении интегрирования. Могут быть заданы следующие параметры:

- *param*(1) = HINIT - начальный размер шага; всегда неотрицателен; по умолчанию равен 0.001 * | *tend* - t_0 |;
- *param*(2) = HMIN - минимальный размер шага; по умолчанию равен 0.0;
- *param*(3) = HMAX - максимальный размер шага; по умолчанию не ограничен - точнее, не может быть более HUGE(HMAX);
- *param*(4) = MXSTEP - максимально допустимое число шагов; по умолчанию равно 500;
- *param*(5) = MXFCN - максимально допустимое число оценок функции; по умолчанию ограничений на число оценок нет;
- *param*(6) = MAXORD - максимальный порядок метода. По умолчанию равен 12, если используется метод Адамса - Мултона, и пяти - в случае метода Гира дифференцирования назад. Заданные по умолчанию значения являются максимально допустимыми;
- *param*(7) = INTRP1 - если отличен от нуля, то подпрограмма IVPAG возвращает *ido* = 4 (см. ниже комментарий 4). По умолчанию INTRP1 = 0;
- *param*(8) = INTRP2 - если отличен от нуля, то подпрограмма IVPAG возвращает *ido* = 5 после каждого успешного шага и *ido* = 6 после каждого неуспешного шага (см. ниже комментарий 4). По умолчанию INTRP2 = 0;
- *param*(9) = SCALE - мера масштабирования среднего значения нормы матрицы Якоби. По умолчанию SCALE = 1.0;
- *param*(10) = INORM - переключатель, задающий норму для вычисления ошибки e_i - абсолютной величины оценки ошибки определения $y_i(t)$. Формула для вычисления нормы задается значением INORM.

  Если

  INORM = 0, то применяется формула MIN(*абсолютная ошибка, относительная ошибка*) = MAX(e_i / w_i), $i = 1, ..., n$, где w_i = MAX(| $y_i(t)$ |, 1.0);

  INORM = 1, то вычисляется абсолютная ошибка, равная MAX(e_i), $i = 1, ..., n$;

  INORM = 2, то используется формула - MAX(e_i / w_i), $i = 1, ..., n$, где w_i = MAX(| $y_i(t)$ |, FLOOR), FLOOR - это PARAM(11);

  INORM = 3, то употребляется масштабированная евклидова норма, определяемая как $ymax = \sqrt{\sum_{i=1}^{n} e_i^2 / w_i^2}$, где w_i = MAX(| $y_i(t)$ |, 1.0).

  Значение по умолчанию INORM = 0;

- *param*(11) = FLOOR - используется при вычислении нормы, когда INORM = 2. Значение по умолчанию - 1.0;
- *param*(12) = METH - метод интегрирования. Если METH = 1, то выбирается метод Адамса - Мултона; при METH = 2 используется метод Гира дифференцирования назад. Значение по умолчанию METH = 1;
- *param*(13) = MITER - индикатор нелинейного решателя. Если задача является жесткой, то наиболее эффективен метод хорд (модифицированный метод Ньютона); поэтому следует задавать MITER = 1 или MITER = 2. Значение

  MITER = 0 задает метод функциональных итераций. Величина IATYPE должна быть задана равной нулю;

  MITER = 1 задает метод хорд с определенным пользователем якобианом;

  MITER = 2 задает метод хорд с якобианом разделенных разностей;

  MITER = 3 задает метод хорд с якобианом, замещенным диагональной матрицей, основанной на производной по направлению. Величина IATYPE должна быть задана равной нулю.

  По умолчанию MITER = 0;
- *param*(14) = MTYPE - задает вид матрицы *A* (если таковая используется) и якобиан (если MITER = 1 или MITER = 2). Если применяются и матрица *A* и якобиан, то они должны иметь один вид.

  Значение

  MTYPE = 0 задает полные матрицы;

  MTYPE = 1 задает ленточные матрицы;

  MTYPE = 2 задает симметрические положительно определенные матрицы;

  MTYPE = 3 задает ленточные симметрические положительно определенные матрицы.

  По умолчанию MTYPE = 0;
- *param*(15) = *nlc* - число нижних кодиагоналей; употребляется, когда MTYPE = 1. Значение по умолчанию - 0;
- *param*(16) = *nuc* - число верхних кодиагоналей; употребляется, когда MTYPE = 3. Значение по умолчанию - 0;
- *param*(17) - не используется;
- *param*(18) = *epsj* - относительный допуск, применяемый при вычислении якобиана разделенных разностей. Значение по умолчанию - SQRT(AMACH(4));

- *param*(19) = IATYPE - тип матрицы *A*. Значение IATYPE = 0 подразумевает, что матрица *A* не используется (система явная); IATYPE = 1 задается, когда *A* - постоянная матрица; IATYPE = 2 устанавливается, если *A* зависит от *t*. По умолчанию IATYPE = 0;
- *param*(20) = *Lda* - ведущий размер массива *a*, представляющего матрицу *A*. Используется, если IATYPE ≠ 0.

  По умолчанию *Lda* равняется:

  n, если MTYPE = 0 или MTYPE = 2;

  nuc + *nlc* + 1, если MTYPE = 1;

  nuc + 1, если MTYPE = 3;
- *param*(21) - *param*(30) - не используются.

Следующие элементы *param* определяются программой:

- *param*(31) = HTRIAL - текущий размер шага;
- *param*(32) = HMINC - вычисленное минимально допустимое значение шага;
- *param*(33) = HMAXC - вычисленное максимально допустимое значение шага;
- *param*(34) = NSTEP - число выполненных шагов;
- *param*(35) = NFCN - число вызовов функции;
- *param*(36) = NJE - число оценок якобиана;
- *param*(37) - *param*(50) - не используются.

y - вектор размера *n* зависимых переменных (значений функций). На входе *y* содержит начальные значения, на выходе - приблизительное решение.

Автоматически для решения предоставляется память:

- $4n + nmeth + npw + nipvt$ байт в случае IVPAG;
- $8n + 2nmeth + 2npw + nipvt$ байт в случае DIVPAG;

здесь

nmeth = 13*n*, если METH = 1;

nmeth = 6*n*, если METH = 2;

npw = 2*n* + *npwm* + *npwa*,

где

npwm = 0, если MITER = 0 или MITER = 3;

$npwm = n^2$, если MITER = 1 или MITER = 2 и MTYPE = 0 или MTYPE = 2;

npwm = *n*(2 * *nlc* + *nuc* + 1), если MITER = 1 или MITER = 2 и MTYPE = 1;

npwm = *n*(*nlc* + 1), если MITER = 1 или MITER = 2 и MTYPE = 3;

npwa = 0, если IATYPE = 0;

npwa = n^2, если IATYPE ≠ 0 и MTYPE = 0;

npwa = *n*(2*nlc* + *nuc* + 1), если IATYPE ≠ 0 и MTYPE = 1;

nipvt = *n*, если MITER = 1 или MITER = 2 и MTYPE = 0 или MTYPE = 1;

nipvt = 1 - в противном случае.

Рабочая область и пользовательская подпрограмма вычисления нормы ошибки, если вызывается I2PAG (DI2PAG), предоставляются явно:

```
CALL I2PAG(ido, n, fcn, fcnj, a, t, tend, tol, param, y, ytemp, ymax,        &
     error, save1, save2, pw, ipvt, vnorm)
```

Ни один дополнительный массив не должен изменяться с момента первого вызова с *ido* = 1 до последнего с *ido* = 3.

Дополнительные параметры подпрограммы I2PAG:

Входной: vnorm.

Выходные: ymax, error.

*Рабочая область: ytemp, save*1, *save*2, *pw, ipvt.*

ytemp - вектор размера *nmeth.*

ymax - вектор размера *n*, содержащий максимальные вычисленные к настоящему времени значения *y*.

error - вектор размера *n*, содержащий оценки ошибок для каждого компонента *y*.

*save*1, *save*2 - векторы размера *n*.

pw - вектор размера *npw*; используется как для хранения якобиана, так и в качестве рабочей памяти.

ipvt - вектор размера *n*.

vnorm - подпрограмма, вычисляющая норму ошибки. Подпрограмма либо предоставляется пользователем, либо применяется подпрограмма I3PRK (DI3PRK) библиотеки IMSL. В последнем случае вместо *vnorm* употребляется имя I3PRK (DI3PRK). И в той и в другой ситуации подпрограмма должна иметь атрибут ENTERNAL. Вызов подпрограммы:

```
CALL vnorm(n, v, y, ymax, enorm)
```

Параметры подпрограммы vnorm:

n - число дифференциальных уравнений.

v - вектор размера *n*, содержащий данные, для которых вычисляется норма.

y - массив размера *n*, содержащий значения зависимой переменной $y(t)$.

ymax - вектор размера *n*, содержащий максимальные вычисленные к настоящему времени значения $|y(t)|$.

enorm - норма вектора *v*.

Первые 4 параметра являются *входными*, последний - *выходным*.

Комментарии:

1. Возможные информационные ошибки:

Тип	*Код*	*Описание*
4	1	После нескольких удачных шагов интегрирование приостановлено в результате замера ошибки вычислений
4	2	Использовано максимально допустимое число оценок функции
4	3	Выполнено максимально допустимое число шагов. Задача может быть жесткой
4	4	На следующем шаге $t + h$ будет равно t. Либо слишком мал параметр *tol*, либо задача является жесткой. Если используется метод Адамса - Мултона, то можно переключиться на метод Гира с дифференцированием назад; если используется метод Гира, то такой возврат означает, что задача является чрезмерно жесткой для заданной комбинации метода и значения параметра *tol*
4	5	После нескольких удачных шагов интегрирование приостановлено в результате тестирования глобальной ошибки с использованием допуска *tol*
4	6	Интегрирование остановлено, так как не удалось пройти тест ошибки даже после деления начального шага на 10^9. Возможно, слишком мало значение *tol*
4	7	Интегрирование остановлено, так как не удалось добиться сходимости процесса даже после деления начального шага на 10^9. Возможно, слишком мало значение *tol*
4	8	Значение IATYPE отлично от нуля, а входная матрица A, на которую умножается y', является вырожденной

2. Могут решаться, как явные ($y' = f(t, y)$), так и неявные ($Ay' = f(t, y)$) системы. Если система явная, то задается *param*(19) = 0 и матрица A не адресуется. В случае неявной системы *param*(14) определяет вид матрицы A. Если *param*(19) = 1, то A - матрица с постоянными элементами. Значение A используется при первом вызове с *ido* = 1, а затем сохраняется до тех пор, пока не выполнен вызов IVPAG с *ido* = 3.

3. Если MTYPE > 0, то MITER должен быть равен 1 или 2.

4. Если *param*(7) отличен от нуля, подпрограмма IVPAG возвращает *ido* = 4 и возобновляет вычисления в точке прерывания, если вызвать ее с *ido* = 4. Если *param*(8) отличен от нуля, подпрограмма прервется немедленно после того, как решит, принимать последний шаг или нет. Значение *ido* = 5 возвращается, если подпрограмма IVPAG планирует принять последний шаг, и возвращается *ido* = 6, если шаг отвергается. Пользователь может изменить значение *ido* с 6 на 5, чтобы вызвать принятие шага, который подпрограмма планирует отвергнуть. После прерывания пользователь может проанализировать имеющие отношение к прерываю параметры *ido, htrial, nstep, nfcn, nje, t* и *y*. Вектор *y* содержит только что вычисленные значения $y(t)$.

Описание:

Подпрограмма IVPAG решает систему первого порядка обыкновенных дифференциальных уравнений вида $y' = f(t, y)$ или $Ay' = f(t, y)$ с начальными условиями, где A - квадратная невырожденная матрица порядка n. Доступны два класса неявных линейных многошаговых методов. Первый содержит неявные методы Адамса - Мултона (до 12-го порядка точности), второй - метод, использующий дифференцирование назад (до 5-го порядка точности), который часто называют жестким методом Гира. В обоих случаях, поскольку базовые формулы являются неявными, на каждом шаге должна решаться система нелинейных уравнений. Матрица производных системы имеет вид $L = A + \eta J$, где η - небольшая величина, вычисляемая подпрограммой IVPAG, а J - якобиан. Когда последний используется, то матрица L либо возвращается пользовательской подпрограммой *fcnj*, либо, как это задано по умолчанию, аппроксимируется разделенными разностями. По умолчанию A является единичной матрицей. Матрица L является вещественной и может быть задана как матрица общего вида, ленточная, симметрическая положительно определенная или ленточная симметрическая положительно определенная. По умолчанию L - матрица общего вида.

Пример 1. Уравнения Эйлера, описывающие движение жесткого тела, не подверженному воздействию внешних сил, имеют вид:

$$y'_1 = y_2 y_3, \qquad y_1(0) = 0.0;$$

$$y'_2 = -y_1 y_3, \qquad y_2(0) = 1.0;$$

$$y'_3 = -0.51 y_1 y_2, \qquad y_3(0) = 1.0.$$

Их решением являются эллиптические функции Якоби $y_1(t)$ = sn(t; k), $y_2(t)$ = cn(t; k), $y_3(t)$ = dn(t; k), где k^2 = 0.51. Для решения подпрограмма IVPAG использует заданный по умолчанию метод Адамса - Мултона. Все параметры имеют установленные по умолчанию значения. Последний вызов IVPAG с *ido* = 3 освобождает рабочую область, выделенную при первом вызове IVPAG. Поскольку *param*(13) = 0, применяются функциональные итерации. Поэтому подпрограмма *fcnj* не вызывается. Однако она присутствует в тексте приложения, поскольку это необходимо для функционирования IVPAG.

```
program ivpag1
 use dfimsl
 integer(4), parameter :: n = 3, nparam = 50
 integer(4) :: ido, iend, nout
 real(4) :: a(1,1), param(nparam), t, tend, tol, y(n)
 external fcn, fcnj
 ! Работаем с заданными по умолчанию параметрами
 param = 0.0
 ido = 1                                  ! Инициализация
 t = 0.0
 y(1) = 0.0; y(2) = 1.0; y(3) = 1.0
 tol = 1.0e-6
 ! Вывод заголовка
 write(*, "(11x, 't', 14x, 'y(1)', 11x, 'y(2)', 11x, 'y(3)')")
 iend = 0                                 ! Осуществляем интегрирование
 do while(iend <= 10)
  iend = iend + 1
  tend = iend
  ! Массив a не используется
  call ivpag(ido, n, fcn, fcnj, a, t, tend, tol, param, y)
  if(iend <= 10) then
   write(*, "(4f15.5)") t, y
   if(iend == 10) ido = 3                 ! Последний вызов
  end if
 end do
end program ivpag1

subroutine fcn(n, x, y, yprime)
 integer(4) :: n
 real(4) :: x, y(n), yprime(n)
 yprime(1) = y(2) * y(3)
 yprime(2) = -y(1) * y(3)
 yprime(3) = -0.51 * y(1) * y(2)
end subroutine fcn
```

```
subroutine fcnj(n, x, y, dypdy)
 ! Эта подпрограмма никогда в данном примере не вызывается
 integer(4) :: n
 real(4) :: x, y(n), dypdy(n, *)
end subroutine fcnj
```

Результат:

t	y(1)	y(2)	y(3)
1.00000	0.80220	0.59705	0.81963
2.00000	0.99537	-0.09615	0.70336
3.00000	0.64141	-0.76720	0.88892
4.00000	-0.26961	-0.96296	0.98129
5.00000	-0.91173	-0.41079	0.75899
6.00000	-0.95751	0.28841	0.72967
7.00000	-0.42877	0.90341	0.95197
8.00000	0.51092	0.85963	0.93106
9.00000	0.97567	0.21926	0.71730
10.00000	0.87790	-0.47885	0.77906

Пример 2. Подпрограмма IVPAG использует метод дифференцирования назад для решения задачи из примера 2 для подпрограммы IVPRK. Дифференцирование назад применяется после задания *param*(12) = 2. Модифицированный метод Ньютона (метод хорд) с якобианом разделенных разностей употребляется для решения нелинейных уравнений. Он задается после установки *param*(13) = 2. Выведенное число оценок y' показывает более высокую эффективность IVPAG по сравнению с IVPRK для данной задачи. Заметим, что число оценок может изменяться в зависимости от точности и других арифметических характеристик используемого компьютера.

```
program ivpag2
 use dfimsl
 integer(4), parameter :: mxparm = 50, n = 2
 integer(4), parameter :: mabse = 1, mbdf = 2, msolve = 2
 integer(4) :: ido, istep, nout
 real(4) :: a(1, 1), param(mxparm), t, tend, tol, y(n)
 external fcn, fcnj
 t = 0.0                              ! Начальные условия
 y(1) = 1.0; y(2) = 0.0; tol = 0.001  ! Допуск для оценки ошибки
 ! Работаем с заданными по умолчанию параметрами
 param = 0.0
 param(10) = mabse                    ! Осуществляем контроль абсолютной ошибки
 param(12) = mbdf                     ! Выбираем дифференцирование назад
```

```
! Задаем метод хорд и якобиан с разделенными разностями
param(13) = msolve
! Вывод заголовка
write(*, "(4x, 'istep', 5x, 'time', 9x, 'y1', 11x, 'y2')")
ido = 1                                    ! Инициализация
istep = 0
do while(istep <= 240)
 istep = istep + 24
 tend = istep
 ! Массив a не используется
 call ivpag(ido, n, fcn, fcnj, a, t, tend, tol, param, y)
 if(istep <= 240) then
   write(*, '(i6, 3f12.3)') istep / 24, t, y
   if(istep == 240) ido = 3                ! Для освобождения рабочей памяти
 end if
end do
! Выводим число вызовов подпрограммы fcn
 write(*, "(4x, 'Number of fcn calls with IVPAG =', f6.0)") param(35)
end program ivpag2

subroutine fcn(n, t, y, yprime)
 integer(4) :: n
 real(4) :: t, y(n), yprime(n)
 real(4) :: ak1, ak2, ak3
 save ak1, ak2, ak3
 data ak1, ak2, ak3 / 294.0e0, 3.0e0, 0.01020408e0 /
 yprime(1) = -y(1) - y(1) * y(2) + ak1 * y(2)
 yprime(2) = -ak2 * y(2) + ak3 * (1.0e0-y(2)) * y(1)
end subroutine fcn

subroutine fcnj(n, t, y, dypdy)
 ! Подпрограмма в данном примере не вызывается
 integer(4) :: n
 real(4) :: t, y(n), dypdy(n,*)
end subroutine fcnj
```

Результат:

istep	time	y1	y2
1	24.000	0.689	0.002
2	48.000	0.636	0.002
3	72.000	0.590	0.002
4	96.000	0.550	0.002
5	120.000	0.515	0.002

6	144.000	0.485	0.002
7	168.000	0.458	0.002
8	192.000	0.434	0.001
9	216.000	0.412	0.001
10	240.000	0.392	0.001

Number of fcn calls with IVPAG = 73

Пример 3. Подпрограмма IVPAG применяет метод дифференцирования назад для решения так называемой задачи Робертсона.

$$y_1' = -c_1 y_2 + c_2 y_2 y_3, \qquad y_1(0) = 1.0;$$

$$y_2' = -y_1' - y_3', \qquad y_2(0) = 0.0;$$

$$y_3' = -c_3 y_2^2, \qquad y_3(0) = 0.0;$$

$$c_1 = 0.004,\ c_2 = 10^4,\ c_2 = 3\times10^7,\ 0.0 \le t \le 10.0.$$

Применяется заданный пользователем якобиан. Допуск для абсолютной ошибки должен быть не менее 10^{-5}.

```
program ivpag3
 use dfimsl
integer(4), parameter :: mxparm = 50, n = 3
 integer(4), parameter :: mabse = 1, mbdf = 2, msolve = 1
 integer(4) :: ido, istep, nout
 real(4) :: a(1,1), param(mxparm), t, tend, tol, y(n)
 external fcn, fcnj
 t = 0.0                                  ! Начальные условия
 y(1) = 1.0; y(2) = 0.0; y(3) = 0.0
 tol = 1.0e-5                             ! Допуск для оценки ошибки
 ! Используем заданные по умолчанию установки
 param = 0.0
 ! Будет осуществляться контроль абсолютной ошибки
 param(10) = mabse
 param(12) = mbdf                         ! Выбираем дифференцирование назад
 ! Выбираем метод хорд и пользовательский якобиан
 param(13) = msolve
 ! Вывод заголовка
 write(*, "(4x, 'istep', 5x, 'time', 9x, 'y1', 11x, 'y2', 11x, 'y3')")
 ido = 1                                  ! Интегрирование
 istep = 0
 do while(istep <= 10)
  istep = istep + 1
  tend = istep
```

```
 ! Массив a не используется
 call ivpag(ido, n, fcn, fcnj, a, t, tend, tol, param, y)
 if (istep .le. 10) then
   write(*, '(i6, f12.2, 3f13.5)') istep, t, y
   if (istep == 10) ido = 3                    ! Для освобождения рабочей памяти
 end if
end do
end program ivpag3

subroutine fcn(n, t, y, yprime)
 integer(4) :: n
 real(4) :: t, y(n), yprime(n)
 real(4) :: c1, c2, c3
 save c1, c2, c3
 data c1, c2, c3 / 0.04e0, 1.0e4, 3.0e7 /
 yprime(1) = -c1 * y(1) + c2 * y(2) * y(3)
 yprime(3) = c3 * y(2)**2
 yprime(2) = -yprime(1) - yprime(3)
end subroutine fcn

subroutine fcnj(n, t, y, dypdy)
 integer(4) :: n
 real(4) :: t, y(n), dypdy(n, *)
 real(4) :: c1, c2, c3
 save c1, c2, c3
 data c1, c2, c3 / 0.04e0, 1.0e4, 3.0e7 /
 dypdy(1:n, 1:n) = 0.0
 dypdy(1, 1) = -c1                           ! Вычисляем частные производные
 dypdy(1, 2) = c2 * y(3)
 dypdy(1, 3) = c2 * y(2)
 dypdy(3, 2) = 2.0 * c3 * y(2)
 dypdy(2, 1) = -dypdy(1, 1)
 dypdy(2, 2) = -dypdy(1, 2) - dypdy(3, 2)
 dypdy(2, 3) = -dypdy(1, 3)
end subroutine fcnj
```

Результат:

istep	time	y1	y2	y3
1	1.00	0.96647	0.00003	0.03350
2	2.00	0.94164	0.00003	0.05834
3	3.00	0.92191	0.00002	0.07806
4	4.00	0.90555	0.00002	0.09443
5	5.00	0.89153	0.00002	0.10845

6	6.00	0.87928	0.00002	0.12070
7	7.00	0.86838	0.00002	0.13160
8	8.00	0.85855	0.00002	0.14143
9	9.00	0.84959	0.00002	0.15039
10	10.00	0.84136	0.00002	0.15862

Пример 4. Решается дифференциальное уравнение в частных производных

$$e^{-t}\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

с начальными условиями

$$u(t = 0, x) = \sin x$$

и граничными условиями

$$u(t, x = 0) = u(t, x = \pi) = 0$$

на прямоугольнике [0, 1]×[0, π]. Используется метод прямых с кусочно-линейной дискредитацией Галеркина.

Точным решением уравнения является функция $u(t, x) = \exp(1 - e^t)\sin x$. При решении интервал [0, π] делится на равные части точками разбиения $x_k = k\pi/(n + 1)$, $k = 0, ..., n + 1$. Неизвестная функция $u(t, x)$ аппроксимируется суммой

$$\sum_{k=1}^{n} c_k(t)\Phi_k(x),$$

где $\Phi_k(x)$ - кусочно-линейная функция, равная единице в x_k и нулю во всех других точках разбиения. Уравнение в частных производных аппроксимируется системой из n обыкновенных дифференциальных уравнений вида

$$A\frac{dc}{dt} = Rc,$$

где A и R - n×n-матрицы.

Матрица A определяется выражением

$$A_{ij} = e^{-t}\int_0^{\pi}\Phi_i(x)\Phi_j(x)dx = \begin{cases} e^{-t}\,2h/3, & \text{если } i = j; \\ e^{-t}\,h/6, & \text{если } i = j \pm 1; \\ 0 \;-\; \text{в противном случае,} \end{cases}$$

где $h = 1/(n + 1)$ - размер сетки. Матрица R задается формулой

$$R_{ij} = \int_0^{\pi} \Phi_i''(x)\Phi_j(x)dx =$$

$$= -\int_0^{\pi} \Phi_i'(x)\Phi_j'(x)dx = \begin{cases} -2/h, & \text{если } i = j; \\ 1/h, & \text{если } i = j \pm 1; \\ 0 & - \text{ в противном случае,} \end{cases}$$

Поскольку система может быть жесткой, применяется метод Гира дифференцирования назад.

В приводимой ниже программе сечение $y(1: n)$ отвечает вектору коэффициентов c. Причем y содержит $n + 2$ элемента: $y(0)$ и $y(n + 1)$ используются для хранения граничных значений. Матрица A зависит от t, поэтому задается *param*(19) = 2. Оценка A выполняется, когда IVPAG завершается с *ido* = 7. Подпрограмма *fcn* вычисляет вектор *Rc*, а подпрограмма *fcnj* - матрицу R. Матрицы A и R представляются как ленточные симметрические положительно определенные, имеющие одну верхнюю кодиагональ.

```
program ivpag4
 use dfimsl
 integer(4), parameter ::n = 9, nparam = 50, nuc = 1, Lda = nuc + 1, nstep = 4
 integer(4) :: i, iatype, ido, imeth, inorm, istep, miter, mtype
 real(4) :: a(lda,n), c, hinit, param(nparam), pi, t, tend,                    &
      tmax, tol, xpoint(0:n + 1), y(0:n + 1)
 character(10) title
 common /comhx/ hx
 real(4) :: hx
 external fcn, fcnj
 ! Задание параметров
 hinit = 1.0e-3; inorm = 1; imeth = 2
 miter = 1; mtype = 3; iatype = 2
 param = 0.0
 param(1) = hinit; param(10) = inorm
 param(12) = imeth; param(13) = miter
 param(14) = mtype; param(16) = nuc
 param(19) = iatype
 pi = const('pi')
 hx = pi / real(n + 1)
 call sset(n - 1, hx / 6., a(1, 2), Lda)
 call sset(n, 2.0 * hx / 3., a(2, 1), Lda)
 do i = 0, n + 1
  xpoint(i) = i * hx
  y(i) = sin(xpoint(i))
```

```
 end do
 tol = 1.0e-6
 t = 0.0
 tmax = 1.0
 ido = 1                                  ! Интегрирование
 istep = 0
 do while(istep <= nstep)
  istep = istep + 1
  tend = tmax * real(istep) / real(nstep)
  do
   call ivpag(ido, n, fcn, fcnj, a, t, tend, tol, param, y(1))
   if(ido /= 7) exit
   ! ido = 7. Задаем матрицу A
   c = exp(-t)
   call sset(n - 1, c * hx / 6., a(1, 2), Lda)
   call sset(n, 2.0 * c * hx / 3., a(2, 1), Lda)
  end do
  if(istep <= nstep) then
  write(title, '(a, f5.3, a)') 'u(t=', t, ')'    ! Вывод решения
  call wrrrn(title, 1, n + 2, y, 1, 0)
  if(istep == nstep) ido = 3              ! Освобождаем рабочую область
  end if
 end do
end program ivpag4

subroutine fcn(n, t, y, yprime)
 use dfimsl
 integer(4) :: n, i
 real(4) :: t, y(*), yprime(n)
 common /comhx/ hx
 real(4) :: hx
 yprime(1) = -2.0 * y(1) + y(2)
 do i = 2, n - 1
  yprime(i) = -2.0 * y(i) + y(i-1) + y(i + 1)
 end do
 yprime(n) = -2.0 * y(n) + y(n - 1)
 call sscal(n, 1.0 / hx, yprime, 1)
end subroutine fcn

subroutine fcnj(n, t, y, dypdy)
 use dfimsl
 integer(4) :: n
 real(4) :: t, y(*), dypdy(2, *)
 common /comhx/ hx
```

```
real(4) :: hx
call sset(n - 1, 1.0 / hx, dypdy(1, 2), 2)
call sset(n, -2.0 / hx, dypdy(2, 1), 2)
end subroutine fcnj
```

Результат:

```
                          u(t = 0.250)
  1       2       3       4       5       6       7       8
0.0000  0.2321  0.4414  0.6076  0.7142  0.7510  0.7142  0.6076

  9      10      11
0.4414  0.2321  0.0000

                          u(t = 0.500)
  1       2       3       4       5       6       7       8
0.0000  0.1607  0.3056  0.4206  0.4945  0.5199  0.4945  0.4206

  9      10      11
0.3056  0.1607  0.0000

                          u(t = 0.750)
  1       2       3       4       5       6       7       8
0.0000  0.1002  0.1906  0.2623  0.3084  0.3243  0.3084  0.2623

  9      10      11
0.1906  0.1002  0.0000

                          u(t = 1.000)
  1       2       3       4       5       6       7       8
0.0000  0.0546  0.1039  0.1431  0.1682  0.1768  0.1682  0.1431

  9      10      11
0.1039  0.0546  0.0000
```

4.3. СИСТЕМЫ АЛГЕБРАИЧЕСКИХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ. ПОДПРОГРАММА DASPG (DDASPG)

Решает систему дифференциальных алгебраических уравнений первого порядка $g(t, y, y') = 0$ при заданных начальных условиях $y(t_0)$ и $y'(t_0)$, используя метод дифференцирования назад Петцольда - Гира. Имеет вызов

CALL DASPG(*n, t, tout, ido, y, ypr, gcn*)

Параметры подпрограммы DASPG:

Пользовательская подпрограмма: gcn.

Входные: n, tout.

Входные/выходные: t, ido, y, ypr.

n - число дифференциальных уравнений.

t - независимая переменная. Перед первым шагом $t = t_0$, т. е. начальному значению независимой переменной.

tout - конечное значение независимой переменной. Этот параметр обновляется, если текущему вызову предшествовал вызов, завершившийся с *ido* = 2.

ido - флаг, характеризующий стадию вычислений. Принимает следующие значения:

- *ido* = 1 - начальный вход;
- *ido* = 2 - нормальный повторный вызов после получения результата;
- *ido* = 3 - последний вызов, приводящий к освобождению рабочей области;
- *ido* = 4 - возврат по причине ошибки.

Пользователь задает *ido* = 1 или *ido* = 3. Иные значения параметра *ido* возвращаются подпрограммой DASPG. Перед первым шагом нужно задать *ido* = 1 и $t = t_0$. Затем подпрограмма установит *ido* = 2, и это значение будет использовано для всех, кроме последнего, вызовов, который выполняется с *ido* = 3. Значения *ido*, большие 4, возникают, только когда вызывается процедура второго уровня D2SPG и используются настройки, связанные с дифференцированием назад.

y - массив размера *n*, содержащий значения зависимых переменных. В этот массив заносятся начальные значения зависимых переменных.

ypr - массив размера *n*, содержащий значения производных *y'*. Также в него заносятся и их начальные значения.

gcn - пользовательская подпрограмма, оценивающая $g(t, y, y')$. Должна обладать атрибутом EXTERNAL. Имеет вызов

CALL *gcn*(*n, t, y, ypr, gval*)

Параметры подпрограммы gcn:

Описание параметров *n, t, y, ypr* см. выше.

gval - массив размера *n*, содержащий значения функции $g(t, y, y')$.

Подпрограмма *gcn* находит значения $y'(t_0)$, такие, что $g(t_0, y, y') = 0$. Пользователь, применяя одну из опций, может указать подпрограмме, что

значение функции g не определено в точке (t, y, y'). Это заставляет подпрограмму уменьшить размер шага или, если это действие окажется нерезультативным, завершить работу.

Регулирование режимов работы подпрограммы DASPG (DDASPG) осуществляется приводимыми в табл. 4.2 и 4.3 целочисленными и вещественными опциями. Подробное описание опций приводится в комментарии 1.

Таблица. 4.2. Целочисленные опции

Значения	*Смысл*
6	Работа с целочисленными опциями
7	Работа с вещественными опциями
IN(1)	Первоначально вызываются DASPG, D2SPG
IN(2)	Скалярный или векторный допуск
IN(3)	Возврат данных на промежуточном шаге
IN(4)	Переход на специальную точку *tstop*
IN(5)	Задаются аналитические формулы для вычисления частных производных
IN(6)	Максимальное число шагов
IN(7)	Управление максимальным размером шага
IN(8)	Управление начальным размером шага
IN(9)	Не используется
IN(10)	Задаются ограничения для зависимых переменных
IN(11)	Согласованные начальные данные
IN(12-15)	Не используются
IN(16)	Число уравнений
IN(17)	Что делает подпрограмма в случае ошибки
IN(18)	Максимальный порядок формул дифференцирования назад
IN(19)	Порядок формулы дифференцирования назад на следующем шаге
IN(20)	Порядок формулы дифференцирования назад на предшествующем шаге
IN(21)	Число шагов
IN(22)	Число оценок функции g
IN(23)	Число оценок матрицы производных
IN(24)	Число неудачных тестов на ошибку
IN(25)	Число неудачных тестов на сходимость

IN(26)	Дифференцирование назад для *g*
IN(27)	Где хранится *g*
IN(28)	Флаг тревоги
IN(29)	Дифференцирование назад для частных производных
IN(30)	Где хранятся частные производные
IN(31)	Дифференцирование назад для решения
IN(32)	Не используется
IN(33)	Где хранятся допуски для вектора
IN(34)	Размещен ли массив для частных производных
IN(35-36)	Проверка размеров пользовательских рабочих массивов
IN(37-50)	Не используются

Таблица 4.3. Вещественные опции

Значения	*Смысл*
INR(1)	Значение *t*
INR(2)	Наибыстрейшее с позиции интегрирования внутреннее значение *t*
INR(3)	Значение *tout*
INR(4)	Точка прекращения интегрирования до *tout*
INR(5)	Значения скаляров *atol* и *rtol*
INR(6)	Начальный размер шага
INR(7)	Максимально допустимый шаг
INR(8)	Обратная величина числа обусловленности
INR(9)	Значение c_j для частных производных
INR(10)	Размер шага на следующей итерации
INR(11)	Размер шага на предшествующей итерации
INR(12-20)	Не используются

Пользователь зачастую может начать употреблять подпрограмму DASPG (DDASPG), не читая сведений о целочисленных и вещественных опциях, поскольку во многих случаях можно обойтись без обращения к ним. Так, в примерах 1 и 2 решаются задачи, в которых задаваемые опциями настройки избыточны.

Автоматически для решения предоставляется память:

- $76 + (maxord + 6)n + (n + k)n(1 - l)$ байт в случае DASPG;
- $117 + 2((maxord + 6)n + (n + k)n(1 - l)) + n$ байт в случае DDASPG.

По умолчанию *maxord* = 5. Значение неотрицательной целочисленной величины *k* задается опцией 16 подпрограммы LSLRG (см. [5], с. 193) через значения *ival*(3) и *ival*(4). По умолчанию $k \le 1$, $l = 0$. Посредством опции IN(34) операнду *l* можно присвоить число 1. Такое уменьшение рабочей области, используемой подпрограммой, предусмотрено для пользователей, которые планируют хранить матрицу с частными производными вне рабочей области памяти и решать самостоятельно линейные алгебраические уравнения, употребляя дифференцирование назад.

Память можно выделить явно, применив D2SPG (DD2SPG):

CALL D2SPG(*n, t, tout, ido, y, ypr, gcn, jgcn, iwk, wk*)

Дополнительные значения параметра ido:

- *ido* = 5 - возврат для вычисления $g(t, y, y')$;
- *ido* = 6 - возврат для оценки матрицы $A = \partial g/\partial y + c_j \partial g/\partial y'$;
- *ido* = 7 - возврат для разложения матрицы $A = \partial g/\partial y + c_j \partial g/\partial y'$;
- *ido* = 8 - возврат для решения линейной системы $A\Delta y = \Delta g$.

Приведенные значения *ido* возникают, лишь когда вызывается подпрограмма второго уровня D2SPG и используются опции, связанные с дифференцированием назад. Для выполнения указанных действий осуществляется повторный вызов подпрограммы D2SPG.

Дополнительные параметры подпрограммы D2SPG:

gcn - имя подпрограммы, вычисляющей значения функции $g(t, y, y')$. По умолчанию применяется подпрограмма, предоставленная пользователем. Когда $g(t, y, y')$ оценивается по формулам дифференцирования назад, в качестве параметра *gcn* может быть употреблена подпрограмма DGSPG (DDGSPG) библиотеки IMSL. В обоих случаях имя подпрограммы должно иметь атрибут EXTERNAL. Техника использования дифференцирования назад для оценки $g(t, y, y')$ демонстрируется в нижеприводимом примере 4.

jgcn - имя подпрограммы, вычисляющей частные производные функции $g(t, y, y')$. Эта подпрограмма может быть предоставлена пользователем. Если частные производные вычисляются с использованием разделенных разностей, то в качестве *jgcn* можно употребить подпрограмму DJSPG (DDJSPG) библиотеки IMSL. Такое положение существует по умолчанию. Если частные производные должны вычисляться явно, то подпрограмма *jgcn* должна быть написана пользователем; также может быть применено дифференцирование назад. С порядком употребления дифференцирования назад для оценки частных производных можно ознакомиться в приводимом ниже примере 4. Заметим: если пользователь в качестве *jgcn* передает имя DJSPG (DDJSPG), то специальной пользовательской подпрограммы для оценки

частных производных не требуется. Информация о том, что частные производные вычисляются по предоставленным пользователем формулам, сообщается целочисленной опцией IN(5) (см. пример 3). Имя подпрограммы, употребляющейся в качестве параметра *jgcn*, должно иметь атрибут EXTERNAL. Подпрограмма *jgcn* имеет вызов

CALL *jgcn*(*n, t, y, ypr, cj, pdg, Ldpdg*)

в котором первые 5 параметров являются *входными*, а 2 последних - *выходными*.

Параметры подпрограммы jgcn:

Описание параметров *n, t, y, ypr* см. выше.

cj - значение c_j, используемое при вычислении частных производных, возвращаемых массивом *pdg*.

pdg - массив размера *Ldpdg*×*n*, содержащий частные производные матрицы $A = \partial g/\partial y + c_j \partial g/\partial y'$; причем $a_{ij} = pdg(i, j)$. Первоначально при вызове подпрограммы *jgcn* массив инициализируется нулями. Далее вычисляются только ненулевые частные производные.

Ldpdg - ведущий размер массива *pdg*; обычно *Ldpdg* = *n*. Ситуация, когда величина *Ldpdg* может быть больше *n*, объясняется в [5, с. 193] при рассмотрении целочисленной опции 16 подпрограммы LSLRG.

Продолжение описания дополнительных параметров подпрограммы D2SPG:

iwk - целочисленный рабочий массив. Его размер равен 35 + *n*.

wk - вещественный рабочий массив, обладающий применяемой точностью. Его размер равен 41 + (*maxord* + 6)*n* + (*n* + *k*) *n*(1 - *l*), где *k* определяется значениями *ival*(3) и *ival*(4) целочисленной опции 16 подпрограммы LSLRG. Значение *l* = 0, если только опция IN(34) не используется для того, чтобы предотвратить размещение матрицы, содержащей частные производные. При употреблении этой опции значение *l* = 1.

Замечание. Содержимое массивов *iwk* и *wk* не должно изменяться с момента первого вызова с *ido* = 1 до последнего с *ido* = 3.

Комментарии:

1. С подпрограммой DASPG используется большое число целых и вещественных опций, управление которыми осуществляется подпрограммами IUMAG, SUMAG и DUMAG. Массив значений целочисленных опций возвращается, когда *iopt*(1) = 6, а вещественных, когда *iopt*(1) = 7. Первоначально обычно запрашиваются значения IN(*i*), *i* = 1, 50 и INR(*i*), *i* = 1, 20 (см. пример 4). По умолчанию IN(*i*) = *i* + 50, *i* = 1, 50 и INR(*i*) = *i* + 50, *i* = 1, 20. Описания опций приводятся в табл. 4.4 и 4.5.

Таблица 4.4. Описание целочисленных опций

Опции	*Описание*
IN(1)	Первый вызов подпрограммы DASPG или D2SPG. Значение 0 устанавливается для первого вызова, 1 - для последующих. Задание *ido* = 1 восстанавливает заданное по умолчанию значение опции. Значение по умолчанию - 0
IN(2)	Флаг, управляющий видом допуска, используемого для решения. Нуль устанавливается, когда употребляются скалярные значения абсолютного и относительного допусков, применяемых для всех компонентов. Если же задаются массивы названных допусков, то в IN(2) записывается единица. В этом случае опция IN(33) используется для получения смещения в векторе *wk*, содержащем 2*n* величин. Причем первоначально в него заносятся абсолютные, а затем относительные значения допусков. Значение по умолчанию - 0
IN(3)	Флаг, определяющий, когда в качестве выходных значений возвращаются *y* и *y'*. Если флаг равен нулю, то *y* и *y'* возвращаются только при *t* = *tout*. Если - единице, то *y* и *y'* возвращаются при каждом внутреннем шаге. Значение по умолчанию - 0
IN(4)	Флаг, определяющий, нужно ли интегрировать после специальной точки *tstop* и выполнять интерполяцию, чтобы получить значения *y* и *y'* в точке *tout*. Если IN(4) = 0, то эти действия разрешены. Если IN(4) = 1, то программа либо считает, что направление интегрирование изменяется на противоположное, либо что в этой точке *y* и *y'* не определены. При IN(4) = 1 программа приближается к *tstop* в направлении интегрирования. Значение *tstop* устанавливается опцией INR(4). По умолчанию IN(4) = 0
IN(5)	Флаг, определяющий, будут ли частные производные вычисляться с использованием односторонних разделенных разностей или они будут вычисляться по предоставленным пользователем формулам. Если он равен нулю, то используются разделенные разности, если единице - пользовательские формулы (см. пример 3). Значение по умолчанию - 0
IN(6)	Максимально допустимое число шагов. Значение по умолчанию - 500
IN(7)	Флаг, определяющий, каким образом устанавливается ограничение на максимальный размер шага. Если он равен нулю, то подпрограмма выбирает максимум самостоятельно; если - единице, то максимум должен быть задан пользователем. Тогда величина максимального размера шага устанавливается опцией INR(7). По умолчанию IN(7) = 0
IN(8)	Флаг, управляющий начальным размером шага. Если он равен нулю, то подпрограмма выбирает начальный шаг самостоятельно; если - единице, то начальный шаг должен быть задан пользователем. Тогда величина начального размера шага устанавливается опцией INR(6). По умолчанию IN(8) = 0

IN(9)	Не используется. Значение по умолчанию - 0
IN(10)	Флаг, управляющий попытками ограничить все компоненты решения неотрицательными значениями. Если он равен нулю, то никаких ограничений на значения компонентов не накладывается; если - единице, то выполняются попытки реализовать имеющиеся ограничения. Значение по умолчанию - 0
IN(11)	Флаг, определяющий, будут ли начальные значения (t, y, y') согласованными. Если IN(11) = 0, то $g(t, y, y') = 0$ в начальной точке; если IN(11) = 1, то подпрограмма выполнит попытку найти величину y' (в результате решения уравнения), удовлетворяющую этому равенству. Значение по умолчанию IN(11) = 1
IN(12-15)	Не используются. Значение по умолчанию - 0
IN(16)	Число уравнений в системе. Значение по умолчанию - 0
IN(17)	Опция, информирующая о действиях и оценках, выполняемых решателем. Значение по умолчанию - 0. Принимает следующие значения: 1 - выполнен шаг интегрирования. Точка *tout* не достигнута; 2 - интегрирование завершено точно в точке *tstop*; 3 - выполнено интегрирование до точки *tstop* и осуществлена интерполяция y и y' вслед за этой точкой; -1 - выполнено слишком много шагов; -2 - слишком мало значение допуска для оценки ошибки; -3 - допуск для относительной ошибки не может быть удовлетворен; -6 - неоднократно не удается провести тест ошибки на последнем шаге; -7 - решатель уравнений по формулам дифференцирования назад не сходится; -8 - матрица частных производных вырожденная; -10 - слишком много неудачных попыток оценок функций. Это означает, что решатель уравнений по формулам дифференцирования назад не сходится; -11 - число неудачных попыток оценок функций превысило предельное значение; подпрограмма завершит работу; -12 - итерация, производимая для вычисления начального значения y', не сходится; -33 - произошла фатальная ошибка, возможно из-за неверных входных данных
IN(18)	Максимальный порядок формул дифференцирования назад. По умолчанию IN(18) = 5
IN(19)	Порядок точности метода дифференцирования назад, который подпрограмма использует на следующем шаге. Значение по умолчанию - IMACH(5), т. е. наибольшее стандартное целое

IN(20)	Порядок точности метода дифференцирования назад, который использован на последнем шаге. Значение по умолчанию - IMACH(5)
IN(21)	Число предпринятых шагов. Значение по умолчанию - 0
IN(22)	Число оценок функции *g*. Значение по умолчанию - 0
IN(23)	Число оценок матрицы частных производных. Значение по умолчанию - 0
IN(24)	Общее число неудачных тестов ошибки. Значение по умолчанию - 0
IN(25)	Общее число неудачных тестов на сходимость, включающее итерации с вырожденной матрицей. Значение по умолчанию - 0
IN(26)	Если IN(26) = 0, то для оценки *g* будет использовано дифференцирование назад; если IN(26) = 1, то - дифференцирование вперед. Для дифференцировании назад применяйте D2SPG, в этом случае D2SPG завершается с *ido* = 5. Для интегрирования оцените *g*, разместите *g* в массиве *wk* с использованием смещения, возвращаемого опцией IN(27), и повторите вызов D2SPG. По умолчанию IN(26) = 1
IN(27)	Возвращает смещение, применяя которое пользователь должен размещать *g* в массиве *wk*. Значение по умолчанию - IMACH(5)
IN(28)	Флаг тревоги. После оценки *g* полученное значение проверяется. Далее используется значение *g*, если в результате проверки флаг равен нулю. Если флаг равен -1, то подпрограмма уменьшает шаг и, возможно, порядок формул дифференцирования назад. Если значение флага - 2, то подпрограмма возвращает управление пользователю немедленно. Эта опция также используется для информирования о том, что матрица частных производных является вырожденной или плохо обусловленной (см. опцию INR(8)), что определяется на этапе разложения матрицы. Значение по умолчанию - 0
IN(29)	Если IN(29) = 0, то для оценки матрицы частных производных используется дифференцирование назад; если IN(29) = 1, то - дифференцирование вперед. Для дифференцирования назад применяйте D2SPG, в этом случае D2SPG завершается с *ido* = 6. Вычислите матрицу *A* частных производных и повторите вызов подпрограммы. Если дифференцирование вперед применяется с линейным решателем, верните частные производные в массиве *wk*, используя смещение, которое возвращается опцией IN(30). По умолчанию IN(29) = 1
IN(30)	Пользователь записывает матрицу частных производных *A* по столбцам в рабочий массив *wk*, используя значение IN(30) как смещение. Опция 16 подпрограммы LSLRG употребляется здесь, чтобы вычислить размер, отводимый под строку во внутреннем рабочем массиве *wk*, содержащем матрицу *A*. Также, если при дифференцировании назад используется решатель линейных систем, пользователь может выбрать удобный для него вариант хранения матрицы в вызывающей программной единице (см. опции N(31) и IN(34). Значение по умолчанию - IMACH(5)

IN(31)	Если IN(31) = 0, то решение линейной системы $A\Delta y = \Delta g$ находится с помощью дифференцирования назад. Если IN(31) = 1, то используется подпрограмма L2CRG для получения *LU*-разложения матрицы и LFSRG для решения линейной системы. Решение возвращается в рабочем массиве *wk*, в котором также хранится *g*. Величина смещения получается опцией IN(27). В случае дифференцирования назад возврат будет выполнен с *ido* = 7 для получения разложения матрицы *A* и с *ido* = 8 для решения системы. В обоих случаях повторите вызов подпрограммы D2SPG. Если матрица *A* вырожденная или плохо обусловленная, то во время разложения можно "поднять" флаг тревоги (см. опции IN(28) и INR(28)). По умолчанию IN(31) = 1
IN(32)	Не используется. Значение по умолчанию - 0
IN(33)	Векторы со значениями *atol* и *rtol* должны записываться в массив *wk*, используя известное смещение. Подпрограмма D2SPG должна быть вызвана до определения смещения
IN(34)	Флаг, определяющий, каким образом выделяется память под матрицу *A*. Если он равен нулю, то матрица *A* хранится в рабочем массиве *wk*; если - единице, рабочая память под матрицу *A* не выделяется. В этом случае пользователь должен применить дифференцирование назад для оценки матрицы частных производных и затем решить линейную систему $A\Delta y = \Delta g$. Значение по умолчанию - 0
IN(35-36)	Размеры массивов *iwk* и *wk*, размещаемых при вызове D2SPG. Размеры проверяются, если значения опции больше нуля. Опция задается при работе с D2SPG. Значения по умолчанию - (0, 0)

Таблица 4.5. Вещественные опции одинарной или двойной точности

Опция	*Описание*
INR(1)	Величина независимой переменной *t*. По умолчанию INR(1) = AMACH(6), т. е. NaN - не число
INR(2)	Наибольшая величина *t*, достигнутая при интегрировании. По умолчанию INR(2) = AMACH(6)
INR(3)	Текущее значение *tout*. По умолчанию INR(3) = AMACH(6)
INR(4)	Следующая специальная точка *tstop*, расположенная до *tout*. Значение по умолчанию - AMACH(6). Употребляется с опцией IN(4)
INR(5)	Пара скалярных величин *atol* и *rtol*, которые используются как допуски при оценке ошибки вычисления компонентов *y*. Для обеих величин по умолчанию установлено значение SQRT(AMACH(4))
INR(6)	Размер начального шага в подпрограмме DASPG. Значение по умолчанию - AMACH(6)

INR(7)	Максимально допустимый размер шага. Значение по умолчанию - AMACH(2), т. е. наибольшее вещественное число одинарной точности
INR(8)	Величина, обратная числу обусловленности матрицы *A*. Определяется, когда для обновления матрицы производных используется дифференцирование вперед. В результате анализа INR(8) пользователь может, применяя IN(28), "поднять" флаг тревоги и объявить систему вырожденной. Значение по умолчанию - AMACH(6)
INR(9)	Значение c_j, используемое с матрицей частных производных при ее оценке с помощью дифференцирования назад. Значение по умолчанию - AMACH(6)
INR(10)	Размер шага, который будет использован на следующей итерации. Значение по умолчанию - AMACH(6)
INR(11)	Размер шага, использованный на предшествующей итерации. Значение по умолчанию - AMACH(6)

2. В подпрограмме DASPG применяется норма

$$d10pg = \sqrt{n^{-1}\sum_{i=1}^{n}\left(v_i / wt_i\right)^2},$$

измеряющая на каждом шаге размер вычисленной ошибки. Норма возвращается функцией

REAL FUNCTION *d*10*pg*(*n*, *v*, *wt*).

Пользователь может употребить и свою функцию, заменив ею *d*10*pg*.

Описание:

Подпрограмма DASPG находит приблизительное решение системы дифференциальных алгебраических уравнений $g(t, y, y') = 0$ с заданными начальными значениями y и y'. Подпрограмма использует формулы дифференцирования назад, позволяющие решать жесткие системы, и стремится сохранять глобальную ошибку в пределах заданного допуска. Подпрограмма эффективна для жестких систем индекса 1 или 0 (понятие индекса см. в [17]). Хотя в приводимых ниже примерах применяется одинарная точность, в документации рекомендуется при употреблении DASPG использовать двойную точность. Подпрограмма основана на коде процедуры DASSL, приведенном в [44].

Пример 1. Уравнение Ван дер Поля $u'' + \mu(u^2 - 1)\,u' + u = 0$, $\mu > 0$, - это нелинейное обыкновенное дифференциальное уравнение второго порядка. Оно описывает свободные колебания одной из простейших нелинейных колебательных систем - осциллятора Ван дер Поля.

При $\mu = 5$ уравнение интегрируется от $t = 0$ до момента завершения устойчивого предельного цикла $t = 26$. Задаются начальные условия $u(0) = 2$ и $u'(0) = -2/3$. Уравнение сводится к дифференциальной алгебраической системе первого порядка

$$g_1 = y_2 - y_1' = 0;$$
$$g_2 = \left(1 - y_1^2\right) y_2 - \varepsilon\left(y_1 + y_2'\right) = 0,$$

где $\varepsilon = 1/\mu$; $y_1 = u$.

Заметим, что начальное значение для y_2' в приводимом ниже коде таково, что $g_2 \neq 0$ при $t = 0$. Подпрограмма DASPG решает получившуюся систему. При этом нет необходимости изменять заданные по умолчанию значения опций. Набор пар $(u(t_j), u'(t_j))$ вычисляется для 260 значений $t_j = 0.1, 26, 0.1$. Рисунок, содержащий цикл Ван дер Поля, создается ОМ.

```
program daspg1
 use dfimsl
 integer(4), parameter :: n = 2, np = 260
 integer(4) :: istep, nstep
 real(4) :: delt, t, tend, y(n), ypr(n)
 ! Массив, введенный для графического отображения цикла Ван дер Поля
 real(4) :: u_upr(2, np)
 external gcn
 ido = 1                                  ! Начальные данные
 t = 0.0; tend = 26.0
 delt = 0.1
 nstep = tend / delt
 y(1) = 2.0; y(2) = -2.0/3.0              ! Начальные условия для функций
 ypr(1) = y(2); ypr(2) = 0.0              ! Начальные условия для производных
 ! Вывод заголовка
 write (*, "(11x, 't', 14x, 'y(1)', 11x, 'y(2)', 10x, 'y''(1)', 10x, 'y''(2)')")
 istep = 0                                ! Интегрирование
 do while(istep <= nstep)
   istep = istep + 1
   call daspg(n, t, t + delt, ido, y, ypr, gcn)
   ! Сохраняем результат для графического вывода
   u_upr(1, istep) = y(1)
   u_upr(2, istep) = ypr(1)
   if(istep == nstep) ido = 3             ! Освобождаем память
 end do
 write(*, '(5f15.5)') tend, y, ypr
 ! Для вывода цикла Ван дер Поля используем
```

```
 ! режим векторного графа отображателя массивов
 call vGraph(u_upr, nstep)
end program daspg1

subroutine gcn(n, t, y, ypr, gval)
 integer(4) :: n
 real(4) :: t, y(n), ypr(n), gval(n)
 real(4) :: eps
 eps = 0.2
 gval(1) = y(2) - ypr(1)
 gval(2) = (1.0 - y(1)**2) * y(2) - eps * (y(1) + ypr(2))
end subroutine gcn

subroutine vGraph(fun, nvalues)          ! Выводит массив как векторный граф ОМ
 use avdef
 use avviewer
 use dflib
 integer(4) :: nvalues
 real(4) :: fun(2, nvalues)
 integer(4) hv, status, nError
 character(1) :: key
 character(av_max_label_len) :: xLabel = 'y'"
 call faglStartWatch(fun, status)          ! Сообщаем ОМ имя отображаемого массива
 print *, "Starting Array Viewer"
 ! Запуск ОМ с использованием fav-подпрограммы
 call favStartViewer(hv, status)
 if(status /= 0) then
  call favGetErrorNo(hv, nError, status)
  if(nError /= 0) then
   print *, "Array Viewer reports error ", nError
   stop
  end if
 end if
 ! Передаем ОМ данные подлежащего отображению массива
 call favSetArray(hv, fun, status)
 ! Задаем заголовок экземпляра ОМ
 call favSetArrayName(hv, "Van der Pol Cycle", status)
 ! Отображаем массив в виде векторного графа
 call favSetGraphType(hv, VectorGraph, status)
 ! Задаем режим вывода заданных пользователем имен осей координат
 call favSetUseAxisLabel(hv, x_axis, 1, status)
 ! Новое (вместо dim1) имя x-оси координат. Длина переменной, задающей имя оси,
 ! равна AV_MAX_LABEL_LEN
 call favSetAxisLabel(hv, x_axis, xLabel, status)
```

```
! Показываем ОМ на экране
call favShowWindow(hv, av_true, status)
print *, "Press any key to close down the viewer"
key = getcharqq( )
call favEndViewer(hv, status)          ! Закрываем ОМ
call faglEndWatch(fun, status)         ! Освобождаем ресурсы
end subroutine vGraph                  ! Результат приведен на рис. 4.1
```

Результат:

t	y(1)	y(2)	y'(1)	y'(2)
26.00000	1.48681	-0.23236	-0.23372	-0.08176

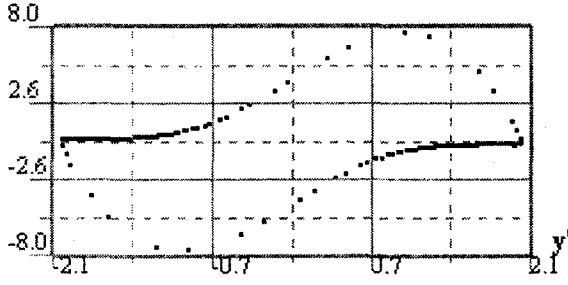


Рис. 4.1. Цикл Ван дер Поля

Пример 2. Уравнения первого порядка движения точечной массы m, подвешенной на невесомой нити длиной l, под действием гравитационной силы mg и натяжения нити λ в декартовой системе координат (p, q) записываются в виде алгебраической системы

$$p' = u;$$

$$q' = v;$$

$$mu' = -p\lambda;$$

$$mv' = -q\lambda - mg;$$

$$p^2 + q^2 + l^2 = 0.$$

В приведенной постановке задача имеет индекс, равный трем. Поэтому она не может быть решена DASPG непосредственно. К сожалению, величину индекса нельзя установить в результате анализа уравнения. Обычно, если индекс больше единицы, в процессе решения возникает ошибка - отсутствие сходимости. В этом примере индекс задачи понижается до единицы в результате двукратного дифференцирования последнего уравнения. Получаемое уравнение

$$m(u^2 + v^2) - mgq - l^2\lambda = 0$$

является уравнением баланса энергии.

Задав начальные условия

$$p(0) = l,\ q(0) = u(0) = v(0) = \lambda(0) = 0$$

и определив зависимые переменные

$$y_1 = p,\ y_2 = q,\ y_3 = u,\ y_4 = v,\ y_5 = \lambda,$$

получим систему

$$g_1 = y_3 - y_1' = 0;$$
$$g_2 = y_4 - y_2' = 0;$$
$$g_3 = -y_1 y_5 - m y_3' = 0;$$
$$g_4 = -y_2 y_5 - mg - m y_4' = 0;$$
$$g_5 = m(y_3^2 + y_4^2) - mgy_2 - l^2 y_5 = 0.$$

В задаче используются английские единицы измерения: фут, фунт. Длина нити равна 6.5 фута, а масса на ее конце составляет 98 фунтов. Поскольку в числовой модели используются единицы СИ, в пользовательской подпрограмме *gcn* выполняется преобразование данных из английских единиц в СИ. Начальные условия соответствуют горизонтальному положению маятника. Они задаются в подпрограмме *gcn* при ее первом вызове с $n = 0$. Максимальная величина натяжения $\lambda(t) = y_5(t)$ вычисляется в выходных точках $t = 0.1, \pi, 0.1$. Экстремальное значение преобразовывается в английские единицы измерения и выводится.

```
program daspg2
 use dfimsl
 integer(4), parameter :: n = 5
 integer(4) :: ido, istep, nstep
 real(4) :: delt, gval(n), maxlb, maxten, t, tend, tmax, y(n), ypr(n)
 external gcn
 ido = 1                                  ! Исходные данные
 t = 0.0
 tend = const('pi')
 delt = 0.1
 nstep = tend / delt
 call gcn(0, t, y, ypr, gval)             ! Вычисляем начальные условия
 istep = 0
 maxten = 0.0
 do while(istep <= nstep)
  istep = istep + 1
```

```
 call daspg(n, t, t + delt, ido, y, ypr, gcn)
 ! Запоминаем максимальное натяжение нити
 if (abs(y(5)) > abs(maxten)) then
  tmax = t
  maxten = y(5)
 end if
 if(istep == nstep) ido = 3
end do
! Переход от кг/с**2 к фунт/с**2
call cunit(maxten, 'kg/s**2', maxlb, 'lb/s**2')
! Выводим значение максимального натяжения нити
print "(' Extreme string tension of', f10.2, ' (lb/s**2)', ' occurred at ', 'time ', f10.2)", maxlb, tmax
end program daspg2

subroutine gcn(n, t, y, ypr, gval)
 use dfimsl
 integer(4) :: n
 real(4) :: t, y(*), ypr(*), gval(*)
 real(4) :: feetl, grav, Lensq, masskg, masslb, meterl, mg
 logical(4) :: first
 save first
 data first / .true. /
 ! Преобразовываем футы и фунты соответственно в метры и килограммы
 if(first) call units_transfer( )
 if(n == 0) then                    ! Определяем начальные условия
  y(1) = meterl                     ! Задаем горизонтальное положение
  y(2:5) = 0.0; ypr(1:5) = 0.0      ! маятника
  return
 end if
 gval(1) = y(3) - ypr(1)            ! Вычисляем невязки
 gval(2) = y(4) - ypr(2)
 gval(3) = -y(1) * y(5) - masskg * ypr(3)
 gval(4) = -y(2) * y(5) - masskg * ypr(4) - mg
 gval(5) = masskg * (y(3)**2 + y(4)**2) - mg * y(2) - Lensq * y(5)

contains

 ! Подпрограмма выполняет преобразование единиц измерения
 subroutine units_transfer( )
  feetl = 6.5
  masslb = 98.0
  ! Переводим футы в метры
  call cunit(feetl, 'ft', meterl, 'meter')
  ! Переводим фунты в килограммы массы
  call cunit(masslb, 'lb', masskg, 'kg')
```

```
  ! Получаем значение ускорения свободного падания
  grav = const('standardgravity')
  mg = masskg * grav
  lensq = meterl**2
  first = .false.
 end subroutine units_transfer
end subroutine gcn
```

Результат:

```
Extreme string tension of 1457.38 (lb/s**2) occurred at time 2.50
```

Пример 3. В примере решается жесткое обыкновенное дифференциальное уравнение из тестового набора [24]. Задача является нелинейной с комплексными собственными значениями. Частные производные вычисляются в пользовательской подпрограмме, имеющей зарезервированное имя DJSPG. Явное вычисление частных производных особенно эффективно, если оценка функции $g(t, y, y')$ требует больших временных затрат. В программе посредством опции IN(5) подпрограмме DASPG сообщается, что матрица производных вычисляется в пользовательской подпрограмме. Для этого вызывается подпрограмма IUMAG, второй параметр которой равен соответствующему номеру главы - числу 10. Также посредством IUMAG задается начальный шаг интегрирования. Допуск, употребляемый при оценке ошибки, изменяется на абсолютный допуск со значением 0.1 * SQRT(AMACH(4)).

```
program daspg3
 use dfimsl
 integer(4), parameter ::n = 4, ichap = 5, iget = 1, inum = 6, iput = 2, irnum = 7
 integer(4) :: ido, in(50), inr(20), iopt(2), ival(2)
 real(4) :: c0, sval(3), t, tend, y(n), ypr(n)
 external gcn
 ido = 1                              ! Начальные данные
 t = 0.0; tend = 1000.0
 c0 = 1.76e-3                         ! Начальные условия
 y(1) = c0; y(2) = 0.0
 y(3) = 0.0; y(4) = 0.0
 ! Начальные значения производных
 ypr(1) = 0.0; ypr(2) = 0.0
 ypr(3) = 0.0; ypr(4) = 0.0
 ! Получаем массив in, содержащий значения целочисленных опций
 iopt(1) = inum
 call iumag('math', ichap, iget, 1, iopt, in)
 ! Получаем массив inr, содержащий значения целочисленных опций
```

```
 iopt(1) = irnum
 call iumag('math', ichap, iget, 1, iopt, inr)
 iopt(1) = inr(6)                    ! Задаем начальный шаг
 sval(1) = 5.0e-5
 iopt(2) = inr(5)                    ! Задаем абсолютный допуск
 sval(2) = 0.1 * sqrt(amach(4))
 sval(3) = 0.0
 call sumag('math', ichap, iput, 2, iopt, sval)
 ! Используем явное вычисление производных
 iopt(1) = in(5)
 ival(1) = 1
 ! Используем заданный в программе начальный шаг
 iopt(2) = in(8)
 ival(2) = 1
 call iumag('math', ichap, iput, 2, iopt, ival)
 ! Вывод заголовка
 write(*, "(11x, 't', 14x, 'y followed by y'")")
 ! Интегрируем заданное уравнение
 call daspg(n, t, tend, ido, y, ypr, gcn)
 write(*, "(f15.5 / (4f15.5))") t, y, ypr
 ! Восстанавливаем заданные по умолчанию значения вещественных опций
 iopt(1) = -inr(5)
 iopt(2) = -inr(6)
 call sumag('math', ichap, iput, 2, iopt, sval)
 ! Восстанавливаем заданные по умолчанию значения целочисленных опций
 iopt(1) = -in(5)
 iopt(2) = -in(8)
 call iumag('math', ichap, iput, 2, iopt, ival)
end program daspg3

subroutine gcn(n, t, y, ypr, gval)      ! Выполняет оценку функции g
 integer(4) :: n
 real(4) :: t, y(n), ypr(n), gval(n)
 real(4) :: c1, c2, c3, c4
 c1 = 7.89e-10; c2 = 1.1e7
 c3 = 1.13e9; c4 = 1.13e3
 gval(1) = -c1 * y(1) - c2 * y(1) * y(3) - ypr(1)
 gval(2) = c1 * y(1) - c3 * y(2) * y(3) - ypr(2)
 gval(3) = c1 * y(1) - c2 * y(1) * y(3) + c4 * y(4) - c3 * y(2) * y(3) - ypr(3)
 gval(4) = c2 * y(1) * y(3) - c4 * y(4) - ypr(4)
end subroutine gcn

! Выполняет оценку матрицы частных производных
subroutine djspg(n, t, y, ypr, cj, pdg, Ldpdg)
```

```
integer(4) :: n, Ldpdg
real(4) :: t, cj, y(n), ypr(n), pdg(Ldpdg, n)
real(4) :: c1, c2, c3, c4
c1 = 7.89e-10; c2 = 1.1e7
c3 = 1.13e9; c4 = 1.13e3
pdg(1, 1) = -c1 - c2 * y(3) - cj
pdg(1, 3) = -c2 * y(1)
pdg(2, 1) = c1
pdg(2, 2) = -c3 * y(3) - cj
pdg(2, 3) = -c3 * y(2)
pdg(3, 1) = c1 - c2 * y(3)
pdg(3, 2) = -c3 * y(3)
pdg(3, 3) = -c2 * y(1) - c3 * y(2) - cj
pdg(3, 4) = c4
pdg(4, 1) = c2 * y(3)
pdg(4, 3) = c2 * y(1)
pdg(4, 4) = -c4 - cj
end subroutine djspg
```

Результат:

```
     t          y followed by y'
 1000.00000
    0.00162     0.00000     0.00000     0.00000
    0.00000     0.00000     0.00000     0.00000
```

Пример 4. Находится решение системы из $n = 10$ обыкновенных дифференциальных уравнений $g = Hy - y'$, где $y(0) = y_0 = (1, 1, ..., 1)^T$. Значение $\sum_{i=1}^{n} y_i(t)$ оценивается при $t = 1$. Элементы независящей от времени нижней матрицы Хессенберга H равны $h_{ij} = \text{MIN}(j - i, 0)$. Для оценки функции g, частных производных, хранимых в матрице $A = \partial g/\partial y + c_j \partial g/\partial y' = H - c_j I$, и решения линейной системы $A\Delta y = \Delta g$ используется дифференцирование назад. Дополнительно с дифференцированием назад частные производные вычисляются по формулам. Рабочая память для матрицы A не выделяется, поскольку она хранится в массиве a, объявляемом в главной программе. Информация о такой организации данных передается подпрограммой IUMAG.

Для разложения матрицы используются преобразования (вращения) Гивенса. Вращения конструируются в процессе вычислений по формулам, приводимым с подпрограммой SROTG (см. [5, разд. 4.4.25]). Подпрограмма D2SPG запоминает элементы c_j, доступ к которым осуществляется при помощи подпрограммы SUMAG. Смещение, в рабочем массиве возвращается

как целочисленная опция, позволяющая определить адрес g и Δg. Вектор решений Δy замещает вектор Δg по этому адресу. В примере демонстрируется механизм получения смещения, необходимого для доступа к области, содержащей g.

Предупреждение. Если пользователь пишет код, предусматривающий вычисление g посредством дифференцирования назад и оценку частных производных при помощи разделенных разностей, то в рабочей области будут существовать два различных отрезка памяти, в которых размещается g.

Также этот пример может служить прототипом для больших (возможно, нелинейных) систем дифференциальных алгебраических уравнений, при решении которых пользователь должен употреблять специальные способы хранения и разложения матрицы A и решения линейной системы $A\Delta y = \Delta g$. В общем случае шаг разложения является подготовительным шагом, результаты которого используются на последующих этапах решения задачи. Разложение, впрочем, может и не выполняться, если система решается итерационными методами.

```
program daspg4
 use dfimsl
integer(4), parameter :: n = 10
integer(4), parameter :: ichap = 5, iget = 1, inum = 6, iput = 2, irnum = 7
integer i, ido, in(50), inr(20), iopt(6), ival(7), iwk(35 + n), j
real a(n, n), gval(n), h(n, n), sc, ss, sumy, sval(1), t,                    &
      tend, wk(41 + 11 * n), y(n), ypr(n), z
external dgspg, djspg
ido = 1                                  ! Начальные данные
t = 0.0e0; tend = 1.0e0
y = 1.0e0                                ! Начальные условия
ypr = 0.0e0
! Инициализация нижней матрицы Хессенберга
h = 0.0e0
do i = 1, n - 1
 do j = 1, i + 1
  h(i, j) = j - i
 end do
end do
do j=1, n
  h(n, j) = j - n
end do
! Получаем массив in, содержащий значения целочисленных опций
iopt(1) = inum
```

```
call iumag('math', ichap, iget, 1, iopt, in)
! Получаем массив inr, содержащий значения вещественных опций
iopt(1) = irnum
call iumag('math', ichap, iget, 1, iopt, inr)
! Задаем дифференцирование назад для оценки g
iopt(1) = in(26); ival(1) = 0
! Установки для вычисления частных производных
iopt(2) = in(5); ival(2) = 1
! Задаем дифференцирование назад для оценки частных производных
iopt(3) = in(29); ival(3) = 0
! Задаем дифференцирование назад для решения линейных уравнений
iopt(4) = in(31); ival(4) = 0
! Отказываемся от выделения памяти для хранения частных производных
iopt(5) = in(34); ival(5) = 1
! Задаем размеры векторов iwk и wk
iopt(6) = in(35); ival(6) = 35 + n; ival(7) = 41 + 11 * n
! Задаем целочисленные опции
call iumag('math', ichap, iput, 6, iopt, ival)
! Вывод заголовка
write(*, "(11x, 't', 6x, 'sum of y(i), i=1, n')")
! Интегрируем систему, используя имеющиеся в IMSL подпрограммы dgspg и djspg
do
 call d2spg(n, t, tend, ido, y, ypr, dgspg, djspg, iwk, wk)
 ! Находим место размещения оценок функции g
 ! В данном примере функция g размещена в памяти однократно
 ! Однако при использовании разделенных разностей для оценки
 ! частных производных создаются две копии g
 iopt(1) = in(27)
 call iumag('math', ichap, iget, 1, iopt, ival)
 ! Анализируем флаг ido
 select case(ido)
 case(1, 4)
  ! Такой возврат не должен выполняться
  write(*, *) ' Unexpected return with ido = ', ido
  call quit( )                          ! Завершаем вычисления
 case(3)
  call quit( )
 case(5)
  ! Был выполнен возврат для оценки g
  call scopy(n, ypr, 1, gval, 1)
  call sgemv('no', n, n, 1.0e0, h, n, y, 1, -1.0e0, gval, 1)
  ! Размещаем g по месту
  call scopy(n, gval, 1, wk(ival(1)), 1)
```

```
 case(6)
  ! Был выполнен возврат для оценки частных производных
  call scopy(n * n, h, 1, a, 1)
  ! Получаем значение c_j, употребляемое при оценке частных производных
  iopt(1) = inr(9)
  call sumag('math', ichap, iget, 1, iopt, sval)
  do i = 1, n                                  ! Вычитаем c_j из диагонали
    a(i, i) = a(i, i) - sval(1)
  end do
 case(7)
  ! Был выполнен возврат для выполнения разложения матрицы A
  do j=1, n - 1
    ! Формируем и применяем вращение Гивенса
    call srotg(a(j, j), a(j, j + 1), sc, ss)
    call srot(n - j, a(j + 1, 1), 1, a(j + 1, j + 1), 1, sc, ss)
  end do
 case(8)
  ! Был выполнен возврат для решения системы
  call scopy(n, wk(ival(1)), 1, gval, 1)
  do j = 1, n - 1
    gval(j) = gval(j) / a(j, j)
    call saxpy(n - j, -gval(j), a(j + 1, j), 1, gval(j + 1), 1)
  end do
  gval(n) = gval(n) / a(n, n)
  ! Реконструируем вращение Гивенса
  do j = n - 1, 1, -1
    z = a(j, j + 1)
    if(abs(z) < 1.0e0) then
      sc = sqrt(1.0e0 - z**2)
      ss = z
    else if(abs(z) > 1.0e0) then
      sc = 1.0e0 / z
      ss = sqrt(1.0e0 - sc**2)
    else
      sc = 0.0e0
      ss = 1.0e0
    end if
    call srot(1, gval(j), 1, gval(j + 1), 1, sc, ss)
  end do
  call scopy(n, gval, 1, wk(ival(1)), 1)
 case(2)
  sumy = sum(y(1:n))
  write(*, "(2f15.5)") tend, sumy
```

```
  ido = 3                              ! Завершаем вычисления
 end select
end do

contains

subroutine quit( )                     ! Подпрограмма, завершающая вычисления
 ! Восстанавливаем заданные по умолчанию значения опций
  in(1:50) = -in(1:50)
  call iumag('math', ichap, iput, 50, in, ival)
  inr(1:20) = -inr(1:20)
  call sumag('math', ichap, iput, 20, inr, sval)
  stop
end subroutine quit
end program daspg4
```

Решение:

```
    t     sum of y(i), i=1, n
 1.00000      65.17057
```

4.4. КРАЕВАЯ ЗАДАЧА

4.4.1. ПОДПРОГРАММА BVPFD (DBVPFD)

Решает краевую задачу - систему дифференциальных уравнений с граничными условиями в двух точках методом конечных разностей с переменным шагом и медленной коррекцией, использующим различные порядки точности. Имеет вызов

CALL BVPFD(*fcneqn, fcnjac, fcnbc, fcnpeq, fcnpbc, n, nleft,* &
ncupbc, tleft, tright, pistep, tol, ninit, tinit, yinit, Ldyini, &
Linear, print, mxgrid, nfinal, tfinal, yfinal, Ldyfin, errest)

Параметры подпрограммы BVPFD:

Пользовательские подпрограммы: fcneqn, fcnjac, fcnbc, fcnpeq, fcnpbc.

Входные: n, nleft, ncupbc, tleft, tright, pistep, tol, ninit, tinit, yinit, Ldyini, Linear, print, mxgrid, Ldyfin.

Выходные: nfinal, tfinal, yfinal, errest.

fcneqn - предоставляемая пользователем подпрограмма, предназначенная для вычисления производных. Имеет вызов

CALL *fcneqn*(*n, t, y, p, dydt*)

fcnjac - предоставляемая пользователем подпрограмма, предназначенная для вычисления якобиана. Имеет вызов

CALL *fcnjac*(*n, t, y, p, dypdy*)

fcnbc - предоставляемая пользователем подпрограмма, предназначенная для оценки граничных условий. Имеет вызов

CALL *fcnbc*(*n, yleft, yright, p, h*)

fcnpeq - предоставляемая пользователем подпрограмма, предназначенная для оценки частной производной y' по параметру p. Имеет вызов

CALL *fcnpeq*(*n, t, y, p, dypdp*)

fcnpbc - предоставляемая пользователем подпрограмма, предназначенная для оценки частных производных граничных условий по параметру p. Имеет вызов

CALL *fcnpbc*(*n, yleft, yright, p, h*)

Параметры пользовательских подпрограмм fcneqn, fcnjac, fcnbc, fcnpbc:

Входные: n, t, y, p, yleft, yright.

Выходные: dydt, dypdp, dypdy, h.

n - число дифференциальных уравнений.

t - независимая переменная.

y - вектор размера *n*, содержащий значения зависимых переменных $y(t)$.

p - параметр продолжения, используемый при решении нелинейных задач (см. комментарий 2).

yleft, yright - векторы размера *n*, содержащие значения зависимой переменной соответственно в левой и правой конечных точках, т. е. в точках *tleft* и *tright*.

dydt - вектор размера *n*, содержащий значения производных $y'(t)$

dypdp - вектор размера *n*, содержащий значения частных производных $a_{ij} = \partial f_i / \partial y_j$, вычисленные в точке (t, y). Значение a_{ij} возвращается как *dypdp*(i, j).

dypdy - массив формы (n, n), содержащий якобиан - частные производные $a_{ij} = \partial f_i / \partial y_j$, вычисленные в точке (t, y). Значение a_{ij} возвращается как *dypdy*(i, j).

h - в подпрограмме *fcnbc* - это вектор размера *n*, содержащий значения невязок для граничных условий. Первоначально $h_i = 0$, $i = 1, ..., n$. Прежде должны быть заданы условия для левой конечной точки, затем задаются условия, включающие обе конечные точки, и в конце - условия для правой конечной точки.

h - в подпрограмме *fcnpbc* это вектор размера *n*, содержащий значения частных производных функций f_i по параметру *p*.

Имена *fcneqn, fcnjac, fcnbc, fcnpeq, fcnpbc* должны быть объявлены в вызывающей программе как EXTERNAL.

Продолжение описания параметров подпрограммы BVPFD:

nleft - число начальных условий; значение *nleft* должно быть больше нуля или равно нулю и меньше, чем *n*.

ncupbc - число связанных граничных условий. Число *nleft* + *ncupbc* должно быть больше нуля или равно нулю и меньше или равно *n*.

tleft, tright - соответственно левая и правая конечные точки.

pistep - начальное значение, используемое для увеличения параметра *p*. Если оно равно нулю, продолжение (см. комментарий 2) не будет использовано при решении задачи и подпрограммы *fcnpeq* и *fcnpbc* вызываться не будут.

tol - допуск, используемый для контроля относительной ошибки. Вычисления останавливаются, когда ABS(*error*(*j*, *i*))/MAX(ABS(*y*(*j*, *i*)), 1.0) < *tol*, *j* = 1, ..., *n* и *i* = 1, ..., *ngrid*. Здесь *error*(*j*, *i*) - вычисляемая для *y*(*j*, *i*) ошибка.

ninit - число начальных точек сетки, включая концевые точки. Должно быть не меньше четырех.

tinit - вектор размера *ninit*, содержащий начальные точки сетки.

yinit - массив формы (*Ldyini*, *ninit*), содержащий *n*×*ninit* начальных предположений о значениях *y* в точках вектора *tinit*. Массив не адресуется, если *ninit* = 0.

Ldyini - ведущий размер *yinit*; обычно *Ldyini* = *n*.

Linear - задается равным .TRUE., если дифференциальные уравнения и граничные условия являются линейными.

print - задается равным .TRUE., если нужно выводить промежуточные результаты.

mxgrid - максимально допустимое число точек сетки.

nfinal - число конечных точек сетки, включающее краевые точки.

tfinal - вектор размера *mxgrid*, содержащий конечные точки сетки. Значимы только первые *nfinal* точек.

yfinal - массив формы (*Ldyfin*, *mxgrid*), содержащий (*n*, *mxgrid*) решений (значений *y*) в точках вектора *tfinal*.

Ldyfin - ведущий размер массива *yfinal*; обычно *Ldyfin* = *n*.

errest - вектор размера *n*. Вектор *errest*(*j*) содержит оценку ошибки в *y*(*j*).

Автоматически для решения предоставляется память:

- $n(3n*mxgrid+4n+1)+mxgrid*(7n+2)+2n*mxgrid+n+mxgrid$ байт в случае BVPFD;
- $2n(3n*mxgrid+4n+1)+2mxgrid(7n+2)+2n*mxgrid+n+mxgrid$ байт в случае DBVPFD.

Память можно выделить явно, употребив B2PFD (DB2PFD):

```
CALL B2PFD(fcneqn, fcnjac, fcnbc, fcnpeq, fcnpbc, n, nleft,          &
     ncupbc, tleft, tright, pistep, tol, ninit, tinit, yinit, Ldyini, &
     Linear, print, mxgrid, nfinal, tfinal, yfinal, Ldyfin,          &
     errest, rwork, iwork)
```

Дополнительные параметры подпрограммы B2PFD:

rwork - вещественный рабочий массив размера $n(3n * mxgrid + 4n + 1) + +mxgrid * (7n + 2)$.

iwork - целочисленный рабочий массив размера $2n*mxgrid+n+mxgrid$.

Комментарии:

1. Возможные информационные ошибки:

Тип	*Код*	*Описание*
4	1	Для решения задачи нужно более чем *mxgrid* точек сетки
4	2	Метод Ньютона расходится
3	3	Метод Ньютона достиг ошибки округления

2. Если значение *pistep* > 0, то подпрограмма BVPFD считает, что пользователь свел задачу к системе с параметром p, имеющей вид:

$$y' = y'(t, y, p);$$

$$h(y_{tleft}, y_{tright}, p) = 0,$$

которая при $p = 0$ является линейной. При $p = 1$ восстанавливается первоначальная задача. Подпрограмма BVPFD автоматически пытается увеличить $p = 0$ до значения $p = 1$, употребляя шаг *pistep*. Величина *pistep* по мере выполнения вычислений начинает увеличиваться. Приращение изменяется подпрограммой BVPFD; нижнее значение приращения - 0.01.

3. Векторы *tinit* и *tfinal* могут совпадать. Также могут совпадать массивы *yinit* и *yfinal*.

Описание:

Подпрограмма BVPFD основана на процедуре PASVA3 [43]. Дискредитация выполняется трапециями на неоднородной сетке. Сетка формируется

адаптивно таким образом, чтобы локальная ошибка была приблизительно одинаковой во всей области. Дискредитация высшего порядка достигается за счет медленной коррекции сетки. В процессе вычислений оценивается глобальная ошибка. Результирующая система нелинейных уравнений решается методом Ньютона с регулируемым шагом. Линеаризованная система уравнений решается специальным методом исключения Гаусса, сохраняющим разреженность матрицы системы.

Пример 1. Решается двухточечная краевая задача

$$y''' - 2y'' + y' - y = \sin t$$

с граничными условиями $y(0) = y(2\pi)$ и $y'(0) = y'(2\pi) = 1$. Ее решение - $y = \sin t$.

Чтобы применить BVPFD, задача приводится к системе обыкновенных дифференциальных уравнений первого порядка путем введения переменных $y_1 = y$, $y_2 = y'$ и $y_3 = y''$. В результате имеем систему

$$y'_1 = y_2, \qquad y_2(0) - 1.0 = 0.0;$$

$$y'_2 = y_3, \qquad y_1(0) - y_1(2\pi) = 0.0;$$

$$y'_3 = 2y_3 - y_2 - y_1 + \sin t, \qquad y_2(2\pi) - 1.0 = 0.0.$$

Заметьте, что задано одно граничное условие в левой конечной точке $t = 0$, одно связывающее граничное условие для левых и правых граничных точек и граничное условие в правой конечной точке. Общее число граничных условий должно совпадать с числом уравнений в системе.

Поскольку параметр *p* не используется при вызове BVPFD, подпрограммы *fcnpeq* и *fcnpbc* не нужны. Следовательно, при вызове BVPFD подпрограммы *fcneqn* и *fcnbc* применяются вместо *fcnpeq* и *fcnpbc*.

```
program bvpfd1
 use dfimsl
 integer(4), parameter :: mxgrid = 45, neqns = 3, ninit = 10,                    &
      Ldyfin = neqns, Ldyini = neqns
 integer(4) :: i, j, ncupbc, nfinal, nleft
 real(4) :: errest(neqns), pistep, tfinal(mxgrid), tinit(ninit), tleft, tol, tright,  &
      yfinal(Ldyfin, mxgrid), yinit(Ldyini, ninit)
 logical(4) :: Linear, print
 external fcnbc, fcneqn, fcnjac
 nleft = 1                                  ! Задаем параметры
 ncupbc = 1
 tol = .001
 tleft = 0.0
 tright = 2.0 * const('pi')
```

```
 pistep = 0.0
 print = .false.
 Linear = .true.
 do i = 1, ninit                          ! Вектор tinit
  tinit(i) = tleft + (i - 1) * (tright - tleft) / float(ninit - 1)
 end do
 yinit = 0.0                              ! Инициализация yinit
 ! Решаем краевую задачу
 call bvpfd(fcneqn, fcnjac, fcnbc, fcneqn, fcnbc, neqns, nleft, ncupbc,                &
       tleft, tright, pistep, tol, ninit, tinit, yinit, Ldyini, Linear, print,         &
       mxgrid, nfinal, tfinal, yfinal, Ldyfin, errest)
 ! Вывод результата
 write(*, "(4x, 'i', 7x, 't', 14x, 'y1', 13x, 'y2', 13x, 'y3')")
 write(*, "(i5, 1p4e15.6)") (i, tfinal(i), (yfinal(j, i), j = 1, neqns), i = 1, nfinal)
 write(*, "(' Error estimates', 4x, 1p3e15.6)") (errest(j), j = 1, neqns)
end program bvpfd1

subroutine fcneqn(neqns, t, y, p, dydx)
 integer(4) :: neqns
 real(4) :: t, p, y(neqns), dydx(neqns)
 ! Задаем дифференциальное уравнение
 dydx(1) = y(2)
 dydx(2) = y(3)
 dydx(3) = 2.0*y(3) - y(2) + y(1) + sin(t)
end subroutine fcneqn

subroutine fcnjac(neqns, t, y, p, dypdy)
 integer(4) :: neqns
 real(4) :: t, p, y(neqns), dypdy(neqns, neqns)
 ! Задаем ∂f_i / ∂y_j
 dypdy(1, 1) = 0.0; dypdy(1, 2) = 1.0; dypdy(1, 3) = 0.0
 dypdy(2, 1) = 0.0; dypdy(2, 2) = 0.0; dypdy(2, 3) = 1.0
 dypdy(3, 1) = 1.0; dypdy(3, 2) = -1.0; dypdy(3, 3) = 2.0
end subroutine fcnjac

subroutine fcnbc(neqns, yleft, yright, p, f)
 integer(4) :: neqns
 real(4) :: p, yleft(neqns), yright(neqns), f(neqns)
 ! Задаем граничные условия
 f(1) = yleft(2) - 1.0
 f(2) = yleft(1) - yright(1)
 f(3) = yright(2) - 1.0
end subroutine fcnbc
```

Результат:

i	t	y1	y2	y3
1	0.000000E+00	-1.123191E-04	1.000000E+00	6.242319E05
2	3.490659E-01	3.419107E-01	9.397087E-01	-3.419580E01
3	6.981317E-01	6.426908E-01	7.660918E-01	-6.427230E-01
4	1.396263E+00	9.847531E-01	1.737333E-01	-9.847453E-01
5	2.094395E+00	8.660529E-01	-4.998747E-01	-8.660057E-01
6	2.792527E+00	3.421830E-01	-9.395474E-01	-3.420648E-01
7	3.490659E+00	-3.417234E-01	-9.396111E-01	3.418948E-01
8	4.188790E+00	-8.656880E-01	-5.000588E-01	8.658733E-01
9	4.886922E+00	-9.845794E-01	1.734571E-01	9.847518E-01
10	5.585054E+00	-6.427721E-01	7.658258E-01	6.429526E-01
11	5.934120E+00	-3.420819E-01	9.395434E-01	3.423986E-01
12	6.283185E+00	-1.123186E-04	1.000000E+00	6.743190E-04

Error estimates 2.840430E-04 1.792939E-04 5.588399E-04

Пример 2. Решается следующая нелинейная задача:

$$y'' - y^3 + (1 + \sin^2 t)\sin t = 0$$

с граничными условиями $y(0) = y(\pi) = 0.0$.

Ее решением является функция $y = \sin t$. Как и в примере 1, исходное уравнение приводится к системе дифференциальных уравнений первого порядка путем введения переменных $y_1 = y$ и $y_2 = y'$. Результирующая система имеет вид:

$$y'_1 = y_2, \qquad y_1(0) = 0.0;$$

$$y'_2 = y_1^3(1 + \sin^2 t)\sin t, \qquad y_1(\pi) = 0.0.$$

В задаче есть одно граничное условие в левой конечной точке и одно граничное условие в правой конечной точке и нет связывающих граничных условий.

Заметим, поскольку параметр *p* не употребляется при вызове BVPFD, подпрограммы *fcnpeq* и *fcnpbc* не нужны. Вместо них используются подпрограммы *fcneqn* и *fcnbc*.

```
program bvpfd2
 use dfimsl
 integer(4), parameter :: mxgrid = 45, neqns = 2, ninit = 12,                &
     Ldyfin = neqns, Ldyini = neqns
```

```
 integer(4) :: i, j, ncupbc, nfinal, nleft
 real(4) :: errest(neqns), pistep, tfinal(mxgrid), tinit(ninit), tleft, tol, tright,          &
      yfinal(Ldyfin, mxgrid), yinit(Ldyini, ninit)
 logical(4) :: Linear, print
 external fcnbc, fcneqn, fcnjac
 nleft = 1                                  ! Задаем параметры
 ncupbc = 0
 tol = 0.001
 tleft = 0.0; tright = const('pi')
 pistep = 0.0
 print = .false.
 Linear = .false.
 do i = 1, ninit                            ! Инициализация tinit и yinit
  tinit(i) = tleft + (i - 1) * (tright - tleft) / float(ninit - 1)
  yinit(1, i) = 0.4 * (tinit(i) - tleft) * (tright - tinit(i))
  yinit(2, i) = 0.4 * (tleft - tinit(i) + tright - tinit(i))
 end do
 ! Решаем краевую задачу
 call bvpfd(fcneqn, fcnjac, fcnbc, fcneqn, fcnbc, neqns, nleft, ncupbc,                       &
      tleft, tright, pistep, tol, ninit, tinit, yinit, Ldyini, Linear, print,                 &
      mxgrid, nfinal, tfinal, yfinal, Ldyfin, errest)
 ! Вывод результата
 write(*, "(4x, 'i', 7x, 't', 14x, 'y1', 13x, 'y2')")
 write(*, "(i5, 1p3e15.6)") (i, tfinal(i), (yfinal(j, i), j = 1, neqns), i = 1, nfinal)
 write(*, "(' Error estimates', 4x, 1p2e15.6)") (errest(j),j=1,neqns)
end program bvpfd2

subroutine fcneqn(neqns, t, y, p, dydt)
 integer(4) :: neqns
 real(4) :: t, p, y(neqns), dydt(neqns)
 ! Задаем дифференциальное уравнение
 dydt(1) = y(2)
 dydt(2) = y(1)**3 - sin(t) * (1.0 + sin(t)**2)
end subroutine fcneqn

subroutine fcnjac(neqns, t, y, p, dypdy)
 integer(4) :: neqns
 real(4) :: t, p, y(neqns), dypdy(neqns, neqns)
 ! Задаем ∂f_i / ∂y_j
 dypdy(1,1) = 0.0; dypdy(1,2) = 1.0
 dypdy(2,1) = 3.0 * y(1)**2; dypdy(2,2) = 0.0
end subroutine fcnjac

subroutine fcnbc(neqns, yleft, yright, p, f)
```

```
integer(4) :: neqns
real(4) :: p, yleft(neqns), yright(neqns), f(neqns)
! Задаем граничные условия
f(1) = yleft(1)
f(2) = yright(1)
end subroutine fcnbc
```

Результат:

i	t	y1	y2
1	0.000000E+00	0.000000E+00	9.999277E-01
2	2.855994E-01	2.817682E-01	9.594315E-01
3	5.711987E-01	5.406458E-01	8.412407E-01
4	8.567980E-01	7.557380E-01	6.548904E-01
5	1.142397E+00	9.096186E-01	4.154530E-01
6	1.427997E+00	9.898143E-01	1.423307E-01
7	1.713596E+00	9.898143E-01	-1.423307E-01
8	1.999195E+00	9.096185E-01	-4.154530E-01
9	2.284795E+00	7.557380E-01	-6.548903E-01
10	2.570394E+00	5.406460E-01	-8.412405E-01
11	2.855994E+00	2.817683E-01	-9.594313E-01
12	3.141593E+00	0.000000E+00	-9.999274E-01
Error estimates	3.906105E-05	7.124186E-05	

Пример 3. Решается нелинейная краевая задача

$$y'' - y^3 = \frac{40}{9}\left(t - \frac{1}{2}\right)^{2/3} - \left(t - \frac{1}{2}\right)^8$$

с граничными условиями $y(0) = y(1) = \pi/2$.

Как и в предшествующих примерах, исходное уравнение приводится к системе дифференциальных уравнений первого порядка. Для этого вводятся переменные $y_1 = y$ и $y_2 = y'$. Результирующая система имеет вид:

$$y'_1 = y_2, \qquad y_1(0) = \pi/2;$$

$$y'_2 = y_1^3 + \frac{40}{9}\left(t - \frac{1}{2}\right)^{2/3} - \left(t - \frac{1}{2}\right)^8, \qquad y_1(1) = \pi/2.$$

Задача путем введения параметра p сводится к семейству задач. При этом второе уравнение системы принимает вид:

$$y_2' = py_1^3 - \frac{40}{9}\left(t - \frac{1}{2}\right)^{2/3} + \left(t - \frac{1}{2}\right)^8. \qquad (4.11)$$

При $p = 0$ задача (4.11) является линейной, а при $p = 1$ осуществляется возврат ко второму уравнению системы.

Производные $\partial y'/\partial p$ задаются в подпрограмме *fcnpeq*, а производные $\partial f/\partial p$ обнуляются в подпрограмме *fcnpbc*.

```
program bvpfd3
 use dfimsl
 integer(4), parameter :: mxgrid = 45, neqns = 2, ninit = 5,                &
      Ldyfin = neqns, Ldyini = neqns
 integer(4) :: i, j, ncupbc, nfinal, nleft
 real(4) :: errest(neqns), pistep, tfinal(mxgrid), tleft, tol,              &
      xrigit, yfinal(Ldyfin, mxgrid)
 real(4) :: tinit(ninit), yinit(Ldyini, ninit)
 external fcnbc, fcneqn, fcnjac, fcnpbc, fcnpeq
 logical(4) :: Linear, print
 data tinit / 0.0, 0.4, 0.5, 0.6, 1.0 /        ! Инициализация
 data ((yinit(i, j), j = 1, ninit), i = 1, neqns) / 0.15749, 0.00215, 0.0,   &
      0.00215, 0.15749, -0.83995, -0.05745, 0.0, 0.05745, 0.83995 /
 nleft = 1                                     ! Задаем параметры
 ncupbc = 0; tol = 0.001
 tleft = 0.0; xright = 1.0
 ! Используются подпрограммы fcnpeq и fcnpbc
 pistep = 0.1
 print = .false.
 Linear = .false.
 ! Решаем краевую задачу
 call bvpfd(fcneqn, fcnjac, fcnbc, fcneqn, fcnbc, neqns, nleft, ncupbc,     &
      tleft, xright, pistep, tol, ninit, tinit, yinit, Ldyini, Linear, print,&
      mxgrid, nfinal, tfinal, yfinal, Ldyfin, errest)
 ! Вывод результата
 write(*, "(4x, 'i', 7x, 't', 14x, 'y1', 13x, 'y2')")
 write(*, "(i5, 1p3e15.6)") (i, tfinal(i), (yfinal(j, i), j = 1, neqns), i = 1, nfinal)
 write(*, "(' Error estimates', 4x, 1p2e15.6)") (errest(j), j = 1, neqns)
end program bvpfd3

subroutine fcneqn(neqns, t, y, p, dydt)
 integer(4) :: neqns
 real(4) :: t, p, y(neqns), dydt(neqns)
 ! Задаем дифференциальное уравнение
 dydt(1) = y(2)
```

```
 dydt(2) = p * y(1)**3 + 40.0 / 9.0 * ((t - 0.5)**2)**(1.0 / 3.0) - (t - 0.5)**8
end subroutine fcneqn

subroutine fcnjac(neqns, t, y, p, dypdy)
 integer(4) :: neqns
 real(4) :: t, p, y(neqns), dypdy(neqns, neqns)
 ! Задаем частные производные ∂f_i / ∂y_j
 dypdy(1, 1) = 0.0; dypdy(1, 2) = 1.0
 dypdy(2, 1) = p * 3.0 * y(1)**2; dypdy(2, 2) = 0.0
end subroutine fcnjac

subroutine fcnbc(neqns, yleft, yright, p, f)
 use dfimsl
 integer(4) :: neqns
 real(4) :: p, yleft(neqns), yright(neqns), f(neqns), pi
 ! Задаем граничные условия
 pi = const('pi')
 f(1) = yleft(1) - pi / 2.0
 f(2) = yright(1) - pi / 2.0
end subroutine fcnbc

subroutine fcnpeq(neqns, t, y, p, dypdp)
 integer(4) :: neqns
 real(4) :: t, p, y(neqns), dypdp(neqns)
 ! Задаем d(dydt)/dp
 dypdp(1) = 0.0
 dypdp(2) = y(1)**3
end subroutine fcnpeq

subroutine fcnpbc(neqns, yleft, yright, p, dfdp)
 integer(4) :: neqns
 real(4) :: p, yleft(neqns), yright(neqns), dfdp(neqns)
 ! Задаем df/dp
 dfdp = 0.0
end subroutine fcnpbc
```

Результат:

i	t	y1	y2
1	0.000000E+00	1.570796E+00	-1.949336E+00
2	4.444445E-02	1.490495E+00	-1.669567E+00
3	8.888889E-02	1.421951E+00	-1.419465E+00
4	1.333333E-01	1.363953E+00	-1.194307E+00
5	2.000000E-01	1.294526E+00	-8.958461E-01
6	2.666667E-01	1.243628E+00	-6.373191E-01

7	3.333334E-01	1.208785E+00	-4.135206E-01
8	4.000000E-01	1.187783E+00	-2.219351E-01
9	4.250000E-01	1.183038E+00	-1.584200E-01
10	4.500000E-01	1.179822E+00	-9.973146E-02
11	4.625000E-01	1.178748E+00	-7.233893E-02
12	4.750000E-01	1.178007E+00	-4.638248E-02
13	4.812500E-01	1.177756E+00	-3.399763E-02
14	4.875000E-01	1.177582E+00	-2.205547E-02
15	4.937500E-01	1.177480E+00	-1.061177E-02
16	5.000000E-01	1.177447E+00	-1.479182E-07
17	5.062500E-01	1.177480E+00	1.061153E-02
18	5.125000E-01	1.177582E+00	2.205518E-02
19	5.187500E-01	1.177756E+00	3.399727E-02
20	5.250000E-01	1.178007E+00	4.638219E-02
21	5.375000E-01	1.178748E+00	7.233876E-02
22	5.500000E-01	1.179822E+00	9.973124E-02
23	5.750000E-01	1.183038E+00	1.584199E-01
24	6.000000E-01	1.187783E+00	2.219350E-01
25	6.666667E-01	1.208786E+00	4.135205E-01
26	7.333333E-01	1.243628E+00	6.373190E-01
27	8.000000E-01	1.294526E+00	8.958461E-01
28	8.666667E-01	1.363953E+00	1.194307E+00
29	9.111111E-01	1.421951E+00	1.419465E+00
30	9.555556E-01	1.490495E+00	1.669566E+00
31	1.000000E+00	1.570796E+00	1.949336E+00
Error estimates		3.448358E-06	5.549869E-05

4.4.2. ПОДПРОГРАММА BVPMS (DBVPMS)

Решает краевую задачу - систему дифференциальных уравнений с граничными условиями в двух точках методом многократной пристрелки. Имеет вызов

CALL BVPMS(*fcneqn, fcnjac, fcnbc, neqns, tleft, tright, dtol, btol, maxit, ninit, tinit, yinit, Ldyini, nmax, nfinal, tfinal, yfinal, Ldyfin*)

Параметры подпрограммы BVPMS:

Пользовательские подпрограммы: fcneqn, fcnjac, fcnbc.

Входные: neqns, tleft, tright, dtol, btol, maxit, ninit, tinit, yinit, Ldyini, nmax, Ldyfin.

Выходные: nfinal, tfinal, yfinal.

Параметры *fcneqn, fcnjac, fcnbc* являются подпрограммами, предоставляемыми пользователем. Их описание дано в предшествующем разделе. Правда, есть отличие: в документации для обозначения числа уравнений вместо имени *n* используется имя *neqns*.

Полученное решение удовлетворяет (в пределах *btol*) условиям $h_i = 0$, $i = 1, ..., neqns$.

Опишем далее те параметры подпрограммы BVPMS, которые не присутствуют в вызове BVPFD. Одноименные параметры этих подпрограмм имеют одинаковый смысл.

dtol - допуск, используемый при вычислении ошибки интегрирования. При решении выполняется попытка управлять локальной ошибкой таким образом, чтобы глобальная ошибка была сравнима с *dtol*.

btol - допуск, используемый при вычислении ошибки граничных условий. Получаемое решение удовлетворяет граничным условиям в пределах допуска *btol*.

maxit - максимально допустимое число ньютоновских итераций. Итерации прекращаются, если сходимость достигнута за меньшее чем *maxit* число итераций. Рекомендуемые значения: *maxit* = 2 для линейных задач и *maxit* = 9 для нелинейных.

ninit - задаваемое пользователем число пристрелочных точек. Может быть равным нулю. Рекомендуемое значение - 10.

tinit - вектор размера *ninit*, содержащий задаваемые пользователем пристрелочные точки. Если *ninit* = 0, то вектор *tinit* не адресуется и подпрограмма выбирает все пристрелочные точки самостоятельно. Однако автоматический выбор пристрелочных точек может потребовать больших вычислительных затрат и его следует использовать при решении линейных задач. Если $ninit \neq 0$, то точки должны задаваться в виде возрастающей последовательности, причем $tinit(1) = tleft$ и $tinit(ninit) = tright$.

nmax - максимально допустимое число пристрелочных точек. Если $ninit \neq 0$, то *nmax* должен быть равен *ninit*. Величина $nmax \geq 2$.

nfinal - результирующее число пристрелочных точек, включая концевые точки.

tfinal - вектор размера *nmax*, содержащий результирующие пристрелочные точки. Значимы только первые *nfinal* точек.

yfinal - массив формы (*Ldyfin*, *nmax*), содержащий *n*×*nmax* решений (значений *y*) в точках вектора *tfinal*.

Автоматически для решения предоставляется память:

- *neqns*(*neqns* + 1)(*nmax* + 12) + 2*neqns* + 30 байт в случае BVPMS;
- 2*neqns*(*neqns* + 1)(*nmax* + 12) + 3*neqns* + 60 байт в случае DBVPMS.

Память можно выделить явно, употребив B2PMS (DB2PMS):

```
CALL B2PMS(fcneqn, fcnjac, fcnbc, neqns, tleft, tright, dtol, btol,    &
     maxit, ninit, tinit, yinit, ldyini, nmax, nfinal, tfinal,         &
     yfinal, ldyfin, work, iwk)
```

Дополнительные параметры подпрограммы B2PMS:

work - вещественный рабочий вектор размера *neqns*(*neqns*+1)(*nmax*+12)+*neqns*+30.

iwk - целочисленный рабочий вектор размера *neqns*.

Комментарии:

1. Возможные информационные ошибки:

Тип	*Код*	*Описание*
1	5	Сходимость достигнута, но для достижения приемлемой точности аппроксимации *y*(*t*) часто необходимо вызвать решатель задачи Коши, например IVPRK, в ближайшей к *tfinal*(*i*) точке с $t \geq tfinal(i)$. Векторы *yfinal*(*j*, :), *j* = 1, ..., *neqns*, используются для задания начальных условий
4	1	Решатель Коши не справился с задачей. Ослабьте допуск *dtol* или см. комментарий 2
4	3	Более чем *nmax* выстрелов необходимо для достижения устойчивости
4	3	Не удалось добиться сходимости за *maxit* ньютоновских итераций. Если задача линейная, выполните дополнительные итерации. Если ошибка сохранилась, проверьте, возвращает ли подпрограмма *fcnjac* верные значения производных. Если с ошибкой не удается справиться, см. комментарий 2
4	4	Решатель линейных систем не справился с задачей. Возможно, что задача не имеет единственного решения или она является существенно нелинейной. В последнем случае см. комментарий 2

2. Многие линейные задачи успешно решаются при автоматическом выборе пристрелочных точек. Нелинейные задачи требуют от пользователя определенных усилий для задания последних. Если подпрограмма не на-

ходит решения, следует увеличить *nmax* или параметризовать задачу. При большом числе пристрелочных точек подпрограмма главным образом применяет конечно-разностный метод, который лучше работает с нелинейностями, чем метод пристрелки. Однако после некоторого предела увеличение числа точек не будет способствовать достижению сходимости. Вопросы параметризации задачи обсуждаются в замечании 4.

3. Если решаемая задача обладает высокой нелинейностью, то для получения сходимости, возможно, потребуется свести задачу к однопараметрической краевой задаче $y' = f(t, y, p)$, $h(y(t_a, t_b, p)) = 0$, в которой при $p = 0$ задача является простой, например линейной, а при $p = 1$ будет выполняться решение исходной задачи. Подпрограмма BVPMS (DBVPMS) автоматически изменяет параметр от $p = 0$ до $p = 1$.
4. Подпрограмма BVPMS рекомендуется для жестких систем дифференциальных уравнений.

Описание:

Пусть $n = neqns$, $m = nfinal$, $t_a = tleft$ и $t_a = tright$. Подпрограмма BVPMS, используя метод многократной пристрелки, решает систему дифференциальных уравнений $y' = f(t, y)$ с граничными условиями

$$h_k(y_1(t_a), ..., y_n(t_a), y_1(t_b), ..., y_n(t_b)) = 0, k = 1, ..., n.$$

Для решения задачи с начальными условиями, возникающей при каждом "выстреле" BVPMS, употребляется модифицированная версия подпрограммы IVPRK. При наличии m пристрелочных точек, включая концевые точки t_a и t_b, решается система из $n*m$ нелинейных уравнений. Решение ищется методом Ньютона, использующим матрицу Якоби почти ленточной структуры. При одном проходе от t_a до t_b во время одной итерации модифицированной подпрограммы IVPRK, работающей с системой из $n(n + 1)$ дифференциальных уравнений, выполняется оценка $n*m$ функций и $n*m \times n*m$-матрицы Якоби. Для большинства задач общий объем вычислений несильно зависит от m.

Многократная пристрелка позволяет удовлетворительно решать большое число плохо обусловленных задач, неразрешимых методами простой пристрелки. С деталями алгоритма можно познакомиться в [49].

Граничные функции должны быть отмасштабированы таким образом, чтобы все компоненты h_k были одного порядка. Это нужно, поскольку при вычислениях контролируется абсолютная ошибка.

Пример. Упругий брус описывают следующие дифференциальные уравнения (см. [55, с. 142-143]):

$$M_{xx} - \frac{NM}{EI} + L(x) = 0;$$
$$EIW_{xx} + M = 0;$$
$$EA_0\left(U_x + W_x^2/2\right) - N = 0;$$
$$N_x = 0,$$

где U - осевое смещение; W - поперечное смещение; N - осевая сила; M - изгибающий момент; E - модуль упругости (модуль Юнга); I - момент инерции; A_0 - площадь поперечного сечения; $L(x)$ - поперечная нагрузка. Пусть имеется фиксированный цилиндрический брус, радиус которого 0.1 дюйма, длина - 10 дюймов, а модуль упругости $E = 10.6\times10^6$ фунт/дюйм^2. Далее, $I = 0.784\times10^{-4}$; $A_0 = \pi/100$ дюйм^2; функция нагрузки $L(x) = -2$, если $3 \le x \le 7$, и равна нулю - в противном случае; граничные условия таковы: $U = W = Wx = 0$ на каждом конце бруса.

Если принять $y_1 = U$, $y_2 = N/EA_0$, $y_3 = W$, $y_4 = W_x$, $y_5 = M/EI$ и $y_6 = M_x/EI$, то вышеприведенные нелинейные уравнения можно записать в виде системы из шести дифференциальных уравнений первого порядка:

$$y_1' = y_2 - 0.5y_4^2;$$
$$y_2' = 0;$$
$$y_3' = y_4;$$
$$y_4' = -y_5;$$
$$y_5' = y_6;$$
$$y_6' = \frac{A_0 y_2 y_5}{I} - \frac{L(x)}{EI}$$

со следующими граничными условиями: $y_1 = y_3 = y_4 = 0$ при $x = 0$ и $x = 10$.

Используем для параметров бруса в нижеприводимой программе следующие обозначения: *a*0 = A_0, *ai* = I. Параметр E будем передавать в процедуры посредством общего блока *param*.

Кроме таблицы результатов выводятся графики, отображающие осевое и поперечное смещение по длине бруса. Для их построения применяется поставляемый с Фортраном отображатель массивов.

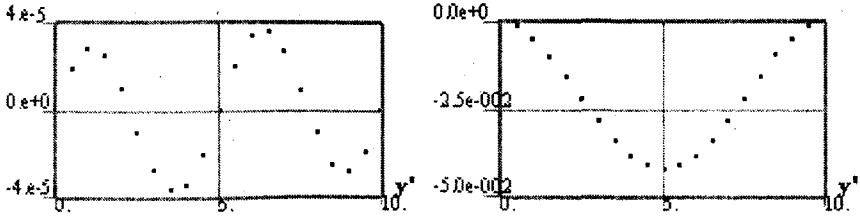
```
program bvpms1
 use dfimsl, nouse => ai                    ! Чтобы избежать дублирования имен
 integer(4), parameter :: neqns = 6, nmax = 21, Ldy = neqns
```

```
integer(4) :: i, maxit, nfinal, ninit
real(4) :: tol, x(nmax), xleft, xright, y(Ldy, nmax)
common /param/ a0, ai, e
real(4) :: a0, ai, e
! Массивы, введенные для вывода графиков осевого и поперечного смещений
real(4), allocatable :: axial(:, :), transverse(:, :)
!dec$attributes array_visualizer :: axial
!dec$attributes array_visualizer :: transverse
external fcnbc, fcneqn, fcnjac
! Параметры бруса
a0 = 3.14e-2                          ! Площадь поперечного сечения
ai = 0.784e-4                         ! Момент инерции
e = 10.6e6                            ! Модуль Юнга
xleft = 0.0; xright = 10.0            ! Параметры подпрограммы BVPMS
tol = 1.0e-4; maxit = 19; ninit = nmax
y = 0.0                               ! Инициализации функции
do i = 1, ninit                       ! Задаем пристрелочные точки
  x(i) = xleft + real(i - 1) / real(ninit - 1) * (xright - xleft)
end do
! Решаем задачу
call bvpms(fcneqn, fcnjac, fcnbc, neqns, xleft, xright, tol, tol, maxit,          &
     ninit, x, y, Ldy, nmax, nfinal, x, y, Ldy)
! Вывод результата
write(*, '(26x, a / 12x, a, 10x, a, 7x, a)') 'Displacement', 'x', 'Axial', 'Transverse'
write(*, '(f15.1, 1p2e15.3)') (x(i), y(1, i), y(3, i), i = 1, nfinal)
! Для вывода графиков осевого и поперечного смещений используем
! режим векторного графа отображателя массивов
allocate(axial(2, nfinal), transverse(2, nfinal))
axial(1, :) = x(1:nfinal)
axial(2, :) = y(1, 1:nfinal)
transverse(1, :) = x(1:nfinal)
transverse(2, :) = y(3, 1:nfinal)
! Текст подпрограммы vGraph приведен в примере 1,
! сопровождающем описание подпрограммы DASPG
call vGraph(axial, nfinal)            ! Осевое смещение
call vGraph(transverse, nfinal)       ! Поперечное смещение
deallocate(axial, transverse)         ! Результат см. на рис. 4.2
end program bvpms1

subroutine fcneqn(neqns, x, y, p, dydx)
integer(4) :: neqns
real(4) :: x, p, y(neqns), dydx(neqns)
real(4) :: force
```

```
 common /param/ a0, ai, e
 real(4) :: a0, ai, e
 force = 0.0                                  ! Задаем производные
 if(x > 3.0 .and. x < 7.0) force = -2.0
 dydx(1) = y(2) - p * 0.5 * y(4)**2
 dydx(2) = 0.0
 dydx(3) = y(4)
 dydx(4) = -y(5)
 dydx(5) = y(6)
 dydx(6) = p * a0 * y(2) * y(5) / ai - force / e / ai
end subroutine fcneqn

subroutine fcnbc(neqns, yleft, yright, p, f)
 integer(4) :: neqns
 real(4) :: p, yleft(neqns), yright(neqns), f(neqns)
 common /param/ a0, ai, e
 real(4) :: a0, ai, e
 ! Задаем граничные условия
 f(1) = yleft(1)
 f(2) = yleft(3)
 f(3) = yleft(4)
 f(4) = yright(1)
 f(5) = yright(3)
 f(6) = yright(4)
end subroutine fcnbc

subroutine fcnjac(neqns, x, y, p, dypdy)
 integer(4) :: neqns
 real(4) :: x, p, y(neqns), dypdy(neqns, neqns)
 common /param/ a0, ai, e
 real(4) :: a0, ai, e
 ! Задаем частные производные d(dydx)/dy
 dypdy = 0.0
 dypdy(1, 2) = 1.0
 dypdy(1, 4) = -p * y(4)
 dypdy(3, 4) = 1.0
 dypdy(4, 5) = -1.0
 dypdy(5, 6) = 1.0
 dypdy(6, 2) = p * y(5) * a0 / ai
 dypdy(6, 5) = p * y(2) * a0 / ai
end subroutine fcnjac
```

Результат:

	Displacement	
x	Axial	Transverse
0.0	1.631E-11	-8.677E-10
5.0	1.914E-05	-1.273E-03
10.0	2.839E-05	-4.697E-03
15.0	2.461E-05	-9.688E-03
20.0	1.008E-05	-1.567E-02
25.0	-9.550E-06	-2.206E-02
30.0	-2.721E-05	-2.830E-02
35.0	-3.644E-05	-3.382E-02
40.0	-3.379E-05	-3.811E-02
45.0	-2.016E-05	-4.083E-02
50.0	-4.414E-08	-4.176E-02
55.0	2.006E-05	-4.082E-02
60.0	3.366E-05	-3.810E-02
65.0	3.627E-05	-3.380E-02
70.0	2.702E-05	-2.828E-02
75.0	9.378E-06	-2.205E-02
80.0	-1.021E-05	-1.565E-02
85.0	-2.468E-05	-9.679E-03
90.0	-2.842E-05	-4.692E-03
95.0	-1.914E-05	-1.271E-03
100.0	0.000E+00	0.000E+00



а *б*

Рис. 4.2. Смещение бруса: а - осевое; б - поперечное

4.5. РЕШЕНИЕ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ В ЧАСТНЫХ ПРОИЗВОДНЫХ. ПОДПРОГРАММА MOLCH (DMOLCH)

Решает, используя метод прямых, систему дифференциальных уравнений вида

$$u_t = f(x, t, u, u_x, u_{xx}).$$

Решение представляется в виде кубических многочленов (сплайнов) Эрмита. Имеет вызов

CALL MOLCH(*ido, fcnut, fcnbc, npdes, t, tend, nx, xbreak, tol, hinit, y, Ldy*)

Параметры подпрограммы MOLCH:

Пользовательские подпрограммы: fcnut, fcnbc.

Входные: npdes, tend, nx, xbreak, tol, hinit, Ldy.

Входные/выходные: ido, t, y.

ido - флаг, характеризующий стадию вычислений. Принимает следующие значения:

- *ido* = 1 - начальный вход (первый вызов);
- *ido* = 2 - нормальный повторный вызов;
- *ido* = 3 - последний вызов, приводящий к освобождению рабочей области;

fcnut - пользовательская подпрограмма, оценивающая функцию u_t. Имеет вызов

CALL *fcnut*(*npdes, x, t, u, ux, uxx, ut*)

Параметры подпрограммы fcnut:

Входные: npdes, x, t, u, ux, uxx.

Выходной: ut.

npdes - число дифференциальных уравнений.

x - пространственная переменная.

t - временная переменная.

u - массив размера *npdes*, содержащий значения зависимых переменных.

ux - массив размера *npdes*, содержащий значения первых производных u_x.

uxx - массив размера *npdes*, содержащий значения вторых производных u_{xx}.

ut - массив размера *npdes*, содержащий значения вычисленных производных u_t.

Имя *fcnut* должно быть объявлено в вызывающей программной единице как EXTERNAL.

fcnbc - пользовательская подпрограмма, оценивающая граничные условия, задаваемые в виде $\alpha_k u_k + \beta_k u_x \equiv \gamma_k$. Пользователь должен задать α_k и β_k, по которым определяются γ_k. Поскольку значения γ_k могут зависеть от t, значения γ'_k также требуются и должны быть предоставлены пользователем. Имеет вызов

CALL *fcnbc*(*npdes, x, t, alpha, beta, gammap*)

Параметры подпрограммы fcnbc:

Входные: npdes, x, t.

Выходные: alpha, beta, gammap.

npdes - число дифференциальных уравнений.

x - пространственная переменная; ее значение определяет вычисляемые граничные условия.

t - временная переменная.

alpha - массив размера *npdes*, содержащий значения α_k.

beta - массив размера *npdes*, содержащий значения β_k.

gammap - массив размера *npdes*, содержащий значения производных $d\gamma_k / dt = \gamma'_k$.

Имя *fcnbc* должно быть объявлено в вызывающей программной единице как EXTERNAL.

Продолжение описания параметров подпрограммы MOLCH:

npdes - число дифференциальных уравнений.

t - независимая переменная. На входе задается начальное значение t_0, на выходе *t* принимает значение, при котором успешно выполнено интегрирование. Обычно новое значение равно *tend*.

tend - значение переменной *t*, при котором следует получить решение.

nx - число точек сетки.

xbreak - массив размера *nx*, содержащий точки разрыва кубических сплайнов Эрмита на оси координат *x*. Значения координат в массиве *xbreak* должны быть записаны в возрастающем порядке. Причем *xbreak*(1) и *xbreak*(*nx*) являются конечными точками отрезка.

tol - допуск для контроля ошибок интегрирования. Выполняется попытка контролировать норму локальной ошибки таким образом, что глобальная ошибка оказывается сравнимой с *tol*.

hinit - начальный шаг интегрирования по *t*. Не может быть задан меньшим нуля. Если *hinit* = 0, то размер начального шага выбирается равным $0.001*|tend - t_0|$.

y - массив формы (*Ldy*, *nx*), содержащий на выходе решение - *npdes*×*nx* значений зависимой переменной *y*. Массив *y* содержит решение, такое, что $y(k, i) = u_x(x, tend)$ при $x = xbreak(i)$. На входе *y* содержит начальные значения. Они должны удовлетворять граничным условиям.

В одном из необязательных приложений MOLCH используются значения производных $u_x(x, t_0)$. Чтобы передать эти значения, пользователь увеличивает размер массива *y* в 2 раза. Необязательные данные о производных вводятся как

$$y(k, i + nx_i) = \frac{\partial u_k(x_i, t_0)}{\partial x_i}.$$

Тогда на выходе массив *y* будет содержать значения производных

$$y(k, i + nx_i) = \frac{\partial u_k(x_i, tend)}{\partial x_i}.$$

Чтобы сообщить, что такие данные предоставляются подпрограмме MOLCH, вызывается управляющая опциями подпрограмма SUMAG (см. примеры 3 и 4).

Ldy - ведущий размер массива *y*; обычно *Ldy* = *npdes*.

Автоматически для решения предоставляется память:

- $2nx*npdes(12npdes^2 + 21npdes + 10)$ байт в случае MOLCH;
- $2nx*npdes(24npdes^2 + 42npdes + 19)$ байт в случае DMOLCH.

Память можно выделить явно, применив M2LCH (DM2LCH):

```
CALL M2LCH(ido, fcnut, fcnbc, npdes, t, tend, nx, xbreak,          &
           tol, hinit, y, Ldy, wk, iwk)
```

Дополнительные параметры подпрограммы M2LCH:

wk - вещественный рабочий массив размера $2nx*npdes(12npdes^2 + 21npdes + 9)$; не должен изменяться между вызовами M2LCH.

iwk - целочисленный рабочий массив размера $2nx*npdes$; не должен изменяться между вызовами M2LCH.

Комментарии:

1. Возможные информационные ошибки:

Тип	*Код*	*Описание*
4	1	После некоторых начальных успешных шагов интегрирование прекращено из-за повторяющихся отрицательных тестов ошибки
4	2	На следующем шаге $x + h$ будет равен x. Причина: либо допуск *tol* чрезмерно мал, либо задача является жесткой
4	3	После некоторых начальных успешных шагов интегрирование прекращено из-за отрицательного теста на *tol*
4	4	Интегрирование прекращено: тест ошибки не удается пройти даже после уменьшения размера шага на 1.0E10.0; возможно, допуск *tol* чрезмерно мал
4	5	Интегрирование прекращено: не удается добиться сходимости процесса даже после уменьшения размера шага на 1.0E10.0; возможно, допуск *tol* чрезмерно мал

2. Опция 11 используется с массивом параметров *param* из 50 элементов (подробно о назначении элементов массива см. в подпрограмме IVPAG). Изменения опции 11 выполняются в результате вызова подпрограммы SUMAG в случае одинарной точности и DUMAG в случае двойной. Некоторые значения массива *param:*

- *param*(1) = *hinit* - начальный шаг интегрирования по *t*;
- *param*(15), *param*(16) - соответственно число нижних и верхних диагоналей, появляющихся в матрице, формируемой при решения по методу Ньютона системы уравнений корректора, использующего формулы дифференцирования назад;
- *param*(17) = 1 - используется для оповещения подпрограммы MOLCH о том, что массив *y* содержит значения производных $u_x(x, t_0)$;
- *param*(31)-*param*(36) - содержат технические данные о процессе интегрирования. Становятся доступными при другом вызове подпрограммы SUMAG (DUMAG), управляющей опциями. Этот вызов выполняется после освобождения памяти, отводимой при работе с MOLCH.

По умолчанию первые 20 элементов *param* имеют значения (0., 0., AMACH(2), 500., 0., 5., 0., 0., 1., 3., 1., 2., 2., 1., AMACH(6), AMACH(6), 0., SQRT(AMACH(4)), 1., 0.). Также по умолчанию *param*(21:50) = AMACH(6).

Описание:

Пусть $m = npdes$, $n = nx$ и $x_i = xbreak(i)$. Подпрограмма MOLCH использует метод прямых для решения системы дифференциальных уравнений в частных производных

$$\frac{\partial u_k}{\partial t} = f_k\left(x,\ t,\ u_1,\ \ldots,\ u_m,\ \frac{\partial u_1}{\partial x},\ \ldots,\ \frac{\partial u_m}{\partial x},\ \frac{\partial^2 u_1}{\partial x^2},\ \ldots,\ \frac{\partial^2 u_m}{\partial x^2}\right)$$

с начальными значениями

$$u_k = u_k(x,\ t) \text{ при } t = t_0$$

и граничными условиями

$$\alpha_k u_k + \beta_k \frac{\partial u_k(a)}{\partial x} = \gamma_k \text{ при } x = x_1 \text{ и } x = x_n \text{ для } k = 1,\ \ldots,\ m.$$

Решение представляется в виде зависящих от x кубических сплайнов Эрмита:

$$\hat{u}_k(x,t) = \sum_{i=1}^{m}\left(a_{ik}(t)\phi_i(x) + b_{ik}(t)\psi_i(x)\right),$$

где $\phi_i(x)$ и $\psi_i(x)$ - стандартные базисные функции для кубических многочленов Эрмита с узлами $x_1 < x_2 < \ldots < x_n$. Составной кубический многочлен имеет непрерывной первую производную. В точках разрыва элементарных кубических кривых выполняются условия:

$$\phi_i(x_l) = \delta_{il},\ \psi_i(x_l) = 0;$$

$$\frac{d\phi_i(x_l)}{dx} = 0,\ \frac{d\psi_i(x_l)}{dx} = \delta_{il}.$$

Согласно методу коллокации коэффициенты аппроксимации получаются таковыми, что пробное решение удовлетворяет на каждом подынтервале дифференциальному уравнению в двух точках (узлах коллокации) Гаусса

$$p_{2j-1} = x_j + \frac{3-\sqrt{3}}{6}\left(x_{j+1} - x_j\right);$$

$$p_{2j} = x_j + \frac{3-\sqrt{3}}{6}\left(x_{j+1} + x_j\right)$$

для $j = 1,\ \ldots,\ n$. Аппроксимация коллокации дифференциального уравнения

$$\frac{da_{ik}}{dt}\phi_i(p_j) + \frac{db_{ik}}{dt}\psi_i(p_j) =$$
$$= f_k\left(p_j, t, \hat{u}_1(p_j), \ldots, \hat{u}_m(p_j),\ \ldots,\ (\hat{u}_1)_{xx}(p_j),\ \ldots,\ (\hat{u}_m)_{xx}(p_j)\right),$$

где $k = 1,\ \ldots,\ m$ и $j = 1,\ \ldots,\ 2(n - 1)$, определяет систему из $2m(n - 1)$ обыкновенных дифференциальных уравнений с $2m*n$ неизвестными коэффициентами функций a_{ik} и b_{ik}. В векторном виде система записывается следующим образом:

$$A\frac{dc}{dt} = F(t, y),$$

где c - вектор коэффициентов размера $2m*n$, $c(t_0) = c_0$, а c_0 - содержит начальные значения коэффициентов. Последние $2m$ уравнений получаются путем дифференцирования граничных условий

$$\alpha_k \frac{da_k}{dt} + \beta_k \frac{db_k}{dt} = \frac{d\gamma_k}{dt}$$

для $k = 1, ..., m$.

Начальные значения $u_k(x, t_0)$ должны удовлетворять граничным условиям. Также функция $\gamma_k(t)$ должна быть непрерывной и иметь гладкую производную, иначе граничные условия не будут надлежащим образом удовлетворяться для $t > t_0$. Если $\alpha_k = \beta_k = 0$, то принимается, что никаких граничных условий для k-й неизвестной в левой конечной точке нет. То же справедливо и для правой конечной точки. Таким образом, в конечных точках коллокация выполнена. Это является полезной чертой для систем дифференциальных уравнений в частных производных первого порядка. Если число дифференциальных уравнений $m = 1$ и число точек разрыва $n = 4$, то

$$A = \begin{pmatrix}
\alpha_1 & \beta_1 & & & & & & \\
\phi_1(p_1) & \psi_1(p_1) & \phi_2(p_1) & \psi_2(p_1) & & & & \\
\phi_1(p_2) & \psi_1(p_2) & \phi_2(p_2) & \psi_2(p_2) & & & & \\
 & & \phi_3(p_3) & \psi_3(p_3) & \phi_4(p_3) & \psi_4(p_3) & & \\
 & & \phi_3(p_4) & \psi_3(p_4) & \phi_4(p_4) & \psi_4(p_4) & & \\
 & & & & \phi_5(p_5) & \psi_5(p_5) & \phi_6(p_5) & \psi_6(p_5) \\
 & & & & \phi_5(p_6) & \psi_5(p_6) & \phi_6(p_6) & \psi_6(p_6) \\
 & & & & & & \alpha_4 & \beta_4
\end{pmatrix}.$$

Вектор

$$c = (a_1, b_1, a_2, b_2, a_3, b_3, a_4, b_4)^T$$

и

$$F = (\gamma'(x_1), f(p_1), f(p_2), f(p_3), f(p_4), f(p_5), f(p_6), \gamma'(x_4))^T.$$

Если $m > 1$, то каждый элемент вышеприведенной матрицы замещается $m \times m$ диагональной матрицей. Так, элемент a_1 замещается на $\mathrm{diag}(a_{11}, ..., a_{1m})$. Элементы a_n, b_1 и b_n замещаются таким же образом. Элементы $\phi_i(p_i)$ и $\psi_i(p_i)$ замещаются $\phi_i(p_i)I_m$ и $\psi_i(p_i)I_m$, где I_m - единичная $m \times m$-матрица (см. [39] о деталях дискретизации ошибок и структуре якобиана).

Входной/выходной массив y содержит значения a_{ki}. Начальные значения b_{ki} возвращаются процедурой CSINT библиотеки IMSL, формирующей кубический сплайн

$$\hat{u}_k(x, t_0),$$

такой, что

$$\hat{u}_k(x_i, t_0) = a_{ki}.$$

Функция CSDER библиотеки IMSL используется для оценки значений

$$\frac{d\hat{u}_k(x_i, t_0)}{dx} \equiv b_{ki}.$$

Существует также необязательный вариант употребления MOLCH, в котором пользователь может передать начальные значения b_{ik}.

Матрица A имеет форму $(2m, n)$ и максимальная ширина ленты матрицы равна $6m - 1$. Структура ленты якобиана матрицы F совпадает со структурой матрицы A. Система решается модифицированной версией подпрограммы IVPAG, в которой, поскольку система обычно является жесткой, по умолчанию применяется метод Гира дифференцирования назад.

Приводимые ниже примеры демонстрируют способы разрешения некоторых проблем, возникающих при употреблении MOLCH. Примеры просты и не охватывают всех возникающих при работе с MOLCH проблем. Два первых примера взяты из [9]. Набор из семи более сложных примеров, некоторые из которых содержат более одного уравнения, дан в [52].

Пример 1. Решается линейное нормализованное уравнение диффузии $u_t = u_{xx}$, $0 \le x \le 1$, $t > t_0$ с начальными значениями $t_0 = 0$, $u(x, t_0) = u_0 = 1$. Задается граничное условие нулевого потока $u_x(1, t) = 0$ $(t > t_0)$ при $x = 1$. Граничное значение $u(0, t)$ ступенчато изменяется от u_0 до $u_1 = 0.1$. Время завершения интегрирования $t = t_\delta = 0.09$.

При таких граничных условиях необходимо снабдить MOLCH значениями производных граничной функции γ' в точках $x = 0$ и $x = 1$. Функция γ при $x = 0$ делает плавный переход от значения u_0 при $t = t_0$ к значению u_1 при $t = t_\delta$. Производная γ' вычисляется в результате оценки кубического интерполяционного многочлена, получаемого функцией CSDER библиотеки IMSL. Интерполяция выполняется в начале пользовательской подпрограммы *fcnbc*. Значения функции и производной $\gamma(t_0) = u_0$, $\gamma'(t_0) = 0$, $\gamma(t_\delta) = u_1$ и $\gamma'(t_\delta) = 0$ подаются на вход процедуры C2HER, возвращающей коэффициенты, используемые CSDER. Заметим, что $\gamma'(t) = 0$, $t > t_\delta$. Функция CSDER не выдает этого значения, поэтому в подпрограмме *fcnbc* существует присваивание $\gamma'(t) = 0$, $t > t_\delta$.

```
program molch1
 use dfimsl
 integer(4), parameter :: npdes = 1, nx = 8, Ldy = npdes
 integer(4) :: i, ido, j, nstep
 real(4) :: hinit, t, tend, tol, xbreak(nx), y(Ldy, nx)
 character(20) :: title
 external fcnbc, fcnut
 ! Задаем точки разрыва и начальные данные
 u0 = 1.0
 do i = 1, nx
  xbreak(i) = float(i - 1) / (nx - 1)
  y(1, i) = u0
 end do
 tol = sqrt(amach(4))                    ! Набор параметров для MOLCH
 hinit = 0.01 * tol; t = 0.0
 ido = 1; nstep = 10
 do j = 1, nstep
  tend = float(j) / float(nstep)
  ! Это позволит увеличить число выводимых точек при малых значениях t,
  ! когда процесс протекает наиболее энергично
  tend = tend**2
  ! Решаем задачу
  call molch(ido, fcnut, fcnbc, npdes, t, tend, nx, xbreak, tol, hinit, y, Ldy)
  ! Формирование заголовка и вывод результата
  write(title, '(a, f4.2)') ' Solution at t =', t
  call wrrrn(title, npdes, nx, y, Ldy, 0)
  ! Освобождаем рабочую область
  if(j == nstep) ido = 3
 end do
end program molch1

subroutine fcnut(npdes, x, t, u, ux, uxx, ut)
 integer(4) :: npdes
 real(4) :: x, t, u(*), ux(*), uxx(*), ut(*)
 ! Задаем уравнение в частных производных
 ut(1) = uxx(1)
end subroutine fcnut

subroutine fcnbc(npdes, x, t, alpha, beta, gamp)
 integer(4) :: npdes
 real(4) :: x, t, alpha(*), beta(*), gamp(*)
 real(4), parameter :: tdelta = 0.09, u0 = 1.0, u1 = 0.1
 integer(4) :: iwk(2), ndata             ! Локальные переменные
 real(4) :: dfdata(2), fdata(2), xdata(2)
```

```
 real(4) :: break(2), cscoef(4, 2)
 logical(4) :: first
 save break, cscoef, first
 data first / .true. /                    ! Возможные действия подпрограммы
 if(first) call prep( )
 ! Задаем граничные условия
 if(x == 0.0) then                        ! x = 0.0
  alpha(1) = 1.0
  beta(1) = 0.0
  gamp(1) = 0.0
  ! Если находимся в граничном слое, то вычисляем ненулевые значения γ'
  if(t <= tdelta) gamp(1) = csder(1, t, 1, break, cscoef)
 else                                     ! x = 1.0
  alpha(1) = 0.0
  beta(1) = 1.0
  gamp(1) = 0.0
end if

contains

 subroutine prep( )
  ! Вычисляем характеристики граничного слоя
  ndata = 2
  xdata(1) = 0.0
  xdata(2) = tdelta
  fdata(1) = u0
  fdata(2) = u1
  dfdata(1) = 0.0
  dfdata(2) = 0.0
  ! Формируем кубический многочлен Эрмита
  call c2her(ndata, xdata, fdata, dfdata, break, cscoef, iwk)
  first = .false.
 end subroutine prep
end subroutine fcnbc
```

Результат:

```
                    Solution at T =0.01
  0.969    0.997    1.000    1.000    1.000    1.000    1.000    1.000
                    Solution at T =0.05
  0.625    0.871    0.963    0.991    0.998    1.000    1.000    1.000
                    Solution at T =0.09
 0.1000   0.4603   0.7171   0.8673   0.9437   0.9781   0.9917   0.9951
```

```
                    Solution at T =0.16
 0.1000   0.3131   0.5072   0.6682   0.7893   0.8709   0.9168   0.9316
                    Solution at T =0.25
 0.1000   0.2568   0.4046   0.5355   0.6429   0.7224   0.7710   0.7874
                    Solution at T =0.36
 0.1000   0.2176   0.3293   0.4292   0.5126   0.5751   0.6139   0.6270
                    Solution at T =0.49
 0.1000   0.1852   0.2661   0.3386   0.3992   0.4448   0.4731   0.4827
                    Solution at T =0.64
 0.1000   0.1588   0.2147   0.2649   0.3067   0.3382   0.3578   0.3644
                    Solution at T =0.81
 0.1000   0.1387   0.1754   0.2084   0.2360   0.2567   0.2696   0.2739
                    Solution at T =1.00
 0.1000   0.1242   0.1472   0.1679   0.1851   0.1981   0.2062   0.2089
```

Пример 2. Решается уравнение нелинейной диффузии с конвекцией и изменяющимися коэффициентами, взятое из [52]. Пример иллюстрирует простой способ программирования процедуры, оценивающей производную u_t. Задача определена следующим образом:

$$u_t = \frac{\partial u}{\partial t} = \frac{\partial\left(D(x)\partial u/\partial x\right)}{\partial x} - \nu(x)\frac{\partial u}{\partial x},\ x \in [0,\ 1],\ t > 0;$$

$$D(x) = \begin{cases} 5, & 0 \le x \le 0.5; \\ 1, & 0.5 < x \le 1.0; \end{cases}$$

$$\nu(x) = \begin{cases} 1000.0, & 0 \le x \le 0.5; \\ 1, & 0.5 < x \le 1.0; \end{cases}$$

$$u(x,\ 0) = \begin{cases} 1, & x = 0; \\ 0, & x > 0.0; \end{cases}$$

$u(0, t) = 1, u(1, t) = 0.$

Кроме таблицы результатов выводится график зависимости $u(x, t = tend)$. Для его построения применяется поставляемый с Фортраном отображатель массивов.

```
program molch2
 use dfimsl
 integer(4), parameter :: npdes = 1, nx = 100, Ldy = npdes
 integer(4) :: i, ido, j, nstep
```

```
real(4) :: hinit, t, tend, tol, xbreak(nx), y(Ldy, nx)
character(20) :: title
! Массив, введенный для вывода графика utend(x) в момент времени t = tend
real(4), allocatable :: uxtend(:, :)
!dec$attributes array_visualizer :: uxtend
external fcnbc, fcnut
u0 = 1.0                                    ! Точки разрыва и начальные значения
do i = 1, nx
 xbreak(i) = float(i - 1) / (nx - 1)
 y(1, i) = 0.
end do
y(1, 1) = u0
! Параметры MOLCH
tol = sqrt(amach(4)); hinit = 0.01 * tol
t = 0.0; ido = 1; nstep = 10
do j = 1, nstep
 tend = float(j) / float(nstep)
 ! Решаем задачу
 call molch(ido, fcnut, fcnbc, npdes, t, tend, nx, xbreak, tol, hinit, y, Ldy)
 if(j == nstep) ido = 3                     ! Освобождаем рабочую память
end do
! Формируем заголовок и выводим результаты
write(title,'(a, f4.2)') ' Solution at t =', t
call wrrrn(title, npdes, nx, y, Ldy, 0)
! Для вывода графиков осевого и поперечного смещений используем
! режим векторного графа отображателя массивов
allocate(uxtend(2, nx))
uxtend(1, :) = xbreak(1:nx)
uxtend(2, :) = y(1, 1:nx)
! Строим график зависимости utend(x) при t = tend
! Текст подпрограммы vGraph приведен в примере 1,
! сопровождающем описание подпрограммы DASPG
call vGraph(uxtend, nx)
deallocate(uxtend)                          ! Результат см. на рис. 4.3
end program molch2

subroutine fcnut(npdes, x, t, u, ux, uxx, ut)
 integer(4) :: npdes
 real(4) :: x, t, u(*), ux(*), uxx(*), ut(*)
 ! Задаем дифференциальное уравнение в частных производных -
 ! нелинейной диффузии с конвекцией и изменяющимися коэффициентами
 if(x <= 0.5) then
  d = 5.0; v = 1000.0
```

```
 else
  d = 1.0; v = 1.0
 end if
 ut(1) = d * uxx(1) - v * ux(1)
end subroutine fcnut

subroutine fcnbc(npdes, x, t, alpha, beta, gamp)
 integer(4) :: npdes
 real(4) :: x, t, alpha(*), beta(*), gamp(*)
 alpha(1) = 1.0
 beta(1) = 0.0
 gamp(1) = 0.0
end subroutine fcnbc
```

Результат:

Solution at t = 1.00

1	2	3	4	5	6	7	8	9	10
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
11	12	13	14	15	16	17	18	19	20
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
21	22	23	24	25	26	27	28	29	30
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
31	32	33	34	35	36	37	38	39	40
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
41	42	43	44	45	46	47	48	49	50
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.997
51	52	53	54	55	56	57	58	59	60
0.984	0.969	0.953	0.937	0.921	0.905	0.888	0.872	0.855	0.838
61	62	63	64	65	66	67	68	69	70
0.821	0.804	0.786	0.769	0.751	0.733	0.715	0.696	0.678	0.659
71	72	73	74	75	76	77	78	79	80
0.640	0.621	0.602	0.582	0.563	0.543	0.523	0.502	0.482	0.461
81	82	83	84	85	86	87	88	89	90
0.440	0.419	0.398	0.376	0.354	0.332	0.310	0.288	0.265	0.242
91	92	93	94	95	96	97	98	99	100
0.219	0.196	0.172	0.148	0.124	0.100	0.075	0.050	0.025	0.000

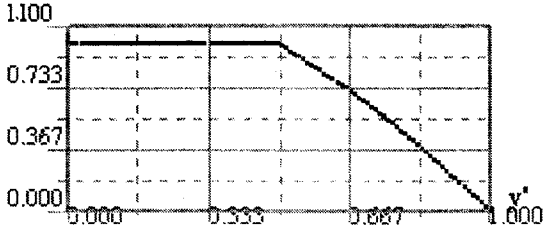


Рис. 4.3. Зависимость u(x) при t = tend

Пример 3. В примере решается линейное нормализованное уравнение диффузии $u_t = u_{xx}$. Однако в отличие от примера 1 здесь дополнительно в качестве начальных данных задаются производные u_x. Из-за ошибки вычисления приближаемых сплайнами производных необходимы их более точные значения, когда начальные данные равны $u(x, 0) = 1 + \cos[(2n - 1)\pi x]$, $n > 1$. На границах поток равен нулю, т. е. $u_x(0, t) = u_x(1, t) = 0$ для $t > 0$. Заметьте, что начальные данные совместимы с этими граничными условиями, поскольку функция производной

$$u_x(x, 0) = \frac{du(x, 0)}{dx} = -(2n-1)\pi \sin\big((2n-1)\pi x\big)$$

равна нулю при $x = 0$ и $x = 1$.

Пример демонстрирует, как использовать подпрограмму SUMAG, управляющую опциями, для установки в массиве *param* значения, информирующего MOLCH о наличии начальных данных о производных u_x. В точках разрыва выходными являются значения $u(x, tend)$ и $u_x(x, tend)$.

```
program molch3
 use dfimsl
 integer(4), parameter :: npdes = 1, nx = 10, Ldy = npdes
 integer(4), parameter :: ichap = 5, iget = 1, iput = 2, kparam = 11
 integer(4) :: i, ido, iopt(1), j, n, nstep
 real(4) :: arg, hinit, param(50), pi, t, tend, tol, xbreak(nx), y(Ldy, 2 * nx)
 character(40) :: title
 external fcnbc, fcnut
 ! Задаем точки разрыва и начальные данные
 n = 5
 pi = const('pi')
 iopt(1) = kparam
 do i = 1, nx
   xbreak(i) = float(i - 1) / (nx - 1)
```

```
  arg = (2.0 * n - 1) * pi
  y(1, i) = 1. + cos(arg * xbreak(i))      ! Значения функции
  y(1, i + nx) = -arg * sin(arg * xbreak(i))  ! Значения первой производной
 end do
 tol = sqrt(amach(4))                      ! Параметры MOLCH
 hinit = 0.01 * tol; t = 0.0
 ido = 1; nstep = 10
 ! Получаем массив param
 call change_opt(iget)
 ! Изменяем значение массива param(17) таким образом, чтобы
 ! указать MOLCH о наличии начальных данных о первой производной
 param(17) = 1.0
 call change_opt(iput)
 ! Получаем выходные данные с шагом 0.001
 tend = 0.
 do j = 1, nstep
  tend = tend + 0.001
  ! Решаем задачу
  call molch(ido, fcnut, fcnbc, npdes, t, tend, nx, xbreak, tol, hinit, y, Ldy)
  ! Формируем заголовок и выводим результат
  write(title, '(a, f5.3)') ' Solution at t =', t
  ! Решение
  call wrtrn(title, npdes, nx, y(:, 1:nx), Ldy, 0)
  ! Для вывода решения и производных следует выполнить вызов
  call wrtrn(title, npdes, 2 * nx, y, Ldy, 0)
  if(j == nstep) ido = 3                   ! Освобождаем рабочую память
 end do
 ! Покажем в качестве примера максимальный размер шага
 call change_opt(iget)
 write(*, *) ' Maximum step size used is: ', param(33)
 ! Возвращаемся к заданным по умолчанию значениям опций
 iopt(1) = -iopt(1)
 call change_opt(iput)
contains                                   ! Подпрограмма, управляющая опциями
 subroutine change_opt(iact)
  integer(4) :: iact
  call sumag('math', ichap, iact, 1, iopt, param)
 end subroutine change_opt
end program molch3
subroutine fcnut(npdes, x, t, u, ux, uxx, ut)
 integer(4) :: npdes
 real(4) :: x, t, u(*), ux(*), uxx(*), ut(*)
```

```
! Задаем уравнение в частных производных
 ut(1) = uxx(1)
end subroutine fcnut

subroutine fcnbc(npdes, x, t, alpha, beta, gamp)
 integer(4) :: npdes
 real(4) :: x, t, alpha(*), beta(*), gamp(*)
 alpha(1) = 0.0
 beta(1) = 1.0
 gamp(1) = 0.0
end subroutine fcnbc
```

Результат:

```
                    Solution at t =0.001
  1.483   0.517   1.483   0.517   1.483   0.517   1.483   0.517   1.483   0.517
                    Solution at t =0.002
  1.233   0.767   1.233   0.767   1.233   0.767   1.233   0.767   1.233   0.767
                    Solution at t =0.003
  1.113   0.887   1.113   0.887   1.113   0.887   1.113   0.887   1.113   0.887
                    Solution at t =0.004
  1.054   0.946   1.054   0.946   1.054   0.946   1.054   0.946   1.054   0.946
                    Solution at t =0.005
  1.026   0.974   1.026   0.974   1.026   0.974   1.026   0.974   1.026   0.974
                    Solution at t =0.006
  1.012   0.988   1.012   0.988   1.012   0.988   1.012   0.988   1.012   0.988
                    Solution at t =0.007
  1.006   0.994   1.006   0.994   1.006   0.994   1.006   0.994   1.006   0.994
                    Solution at t =0.008
  1.003   0.997   1.003   0.997   1.003   0.997   1.003   0.997   1.003   0.997
                    Solution at t =0.009
  1.001   0.999   1.001   0.999   1.001   0.999   1.001   0.999   1.001   0.999
                    Solution at t =0.010
  1.001   0.999   1.001   0.999   1.001   0.999   1.001   0.999   1.001   0.999
Maximum step size used is: 1.0000002E-02
```

Пример 4. В примере решается линейное нормализованное гиперболическое дифференциальное уравнение в частных производных $u_{tt} = u_{xx}$, моделирующее вибрирующую струну. Оно сводится к системе обыкновенных дифференциальных уравнений первого порядка. Для этого вводится новая

зависимая переменная $u_t = v$. Второе уравнение системы таково: $v_t = u_{xx}$. В качестве начальных данных принимаются $u(x, 0) = \sin(\pi x)$ и $u_t(x, 0) = v(x, 0) = 0$. Концы струны фиксированы, поэтому $u(0, t) = u(1, t) = v(0, t) = v(1, t) = 0$. Точное решение задачи известно: $u(x, t) = \sin(\pi x)\cos(\pi t)$. Невязки вычисляются на выходе для $0 < t \le 2$. Результат получается за 200 шагов; размер шага 0.01.

Хотя применение MOLCH дает удовлетворительный результат для этого уравнения, пользователю следует иметь в виду, что в гиперболических системах вследствие нелинейности могут происходить скачки, приводящие к непредсказуемым результатам. Эта проблема обсуждается в [20].

```
program molch4
 use dfimsl
 integer(4), parameter :: npdes = 2, nx = 10, Ldy = npdes
 integer(4), parameter :: ichap = 5, iget = 1, iput = 2, kparam = 11
 integer(4) :: i, iact, ido, iopt(1), j, nstep
 real(4) :: hinit, param(50), pi, t, tend, tol, xbreak(nx), y(Ldy, 2 * nx), error(nx)
 external fcnbc, fcnut
 ! Задаем точки разрыва и начальные данные
 pi = const('pi')
 iopt(1) = kparam
 do i = 1, nx
  xbreak(i) = float(i-1)/(nx-1)
  ! Значения функции
  y(1, i) = sin(pi * xbreak(i))
  y(2, i) = 0.
  ! Значения первой производной
  y(1, i + nx) = pi * cos(pi * xbreak(i))
  y(2, i + nx) = 0.0
 end do
 tol = 0.1 * sqrt(amach(4))                ! Параметры MOLCH
 hinit = 0.01*tol; t = 0.0
 ido = 1; nstep = 200
 ! Получаем массив param
 call change_opt(iget)
 ! Изменяем значение массива param(17) таким образом, чтобы
 ! указать MOLCH о наличии начальных данных о первой производной
 param(17) = 1.0
 call change_opt(iput)
 ! Получаем выходные данные с шагом 0.001 и вычисляем ошибки
 tend = 0.0; erru = 0.0
 do j = 1, nstep
  tend = tend + 0.01
```

```
  ! Решаем задачу
  call molch (ido, fcnut, fcnbc, npdes, t, tend, nx, xbreak, tol, hinit, y, Ldy)
  do i = 1, nx
    error(i) = y(1, i) - sin(pi * xbreak(i)) * cos(pi * tend)
  end do
  do i = 1, nx
    erru = amax1(erru, abs(error(i)))
  end do
  if(j == nstep) ido = 3                    ! Освобождаем рабочую память
end do
 ! Покажем в качестве примера максимальный размер шага
 call change_opt(iget)
 write(*, *) ' Maximum error in u(x, t) divided by TOL: ', erru / tol
 write(*, *) ' Maximum step size used is: ', param(33)
 ! Возвращаемся к заданным по умолчанию значениям опций
 iopt(1) = -iopt(1)
 call change_opt(iput)
contains                                    ! Подпрограмма, управляющая опциями
 subroutine change_opt(iact)
  integer(4) :: iact
  call sumag('math', ichap, iact, 1, iopt, param)
 end subroutine change_opt
end program molch4

subroutine fcnut(npdes, x, t, u, ux, uxx, ut)
 integer(4) :: npdes
 real(4) :: x, t, u(*), ux(*), uxx(*), ut(*)
 ! Задаем дифференциальное уравнение в частных производных
 ut(1) = u(2)
 ut(2) = uxx(1)
end subroutine fcnut

subroutine fcnbc(npdes, x, t, alpha, beta, gamp)
 integer(4) :: npdes
 real(4) :: x, t, alpha(*), beta(*), gamp(*)
 alpha(1) = 1.0; beta(1) = 0.0; gamp(1) = 0.0
 alpha(2) = 1.0; beta(2) = 0.0; gamp(2) = 0.0
end subroutine fcnbc
```

Результат:

```
Maximum error in u(x, t) divided by TOL: 1.263677
Maximum step size used is: 9.999990E-02
```

4.6. РЕШЕНИЕ УРАВНЕНИЙ ПУАССОНА И ГЕЛЬМГОЛЬЦА

4.6.1. ПОДПРОГРАММА FPS2H (DFPS2H)

Решает уравнения Пуассона или Гельмгольца в двумерной прямоугольной области, используя быстрый пуассоновский решатель, основанный на конечно-разностной схеме на однородной сетке. Имеет вызов

CALL FPS2H(*prhs, brhs, coefu, nx, ny, ax, bx, ay, by, ibcty, iorder, u, Ldu*)

Параметры подпрограммы FPS2H:

Пользовательские функции: prhs, brhs.

Входные: coefu, nx, ny, ax, bx, ay, by, ibcty, iorder, Ldu.

Выходной: u.

prhs - пользовательская функция, выполняющая оценку правой части уравнения. Должна иметь атрибут EXTERNAL. Имеет вызов

result = *prhs*(*x, y*)

Параметры функции prhs:

x, y - соответственно *x*- и *y*-координаты области; являются *входными.*

brhs - пользовательская функция, выполняющая оценку граничных условий правой части уравнения. Должна иметь атрибут EXTERNAL. Имеет вызов

resul = *brhs*(*iside, x, y*)

Параметры функции brhs:

iside - номер стороны; см. ниже описание параметра *ibcty.*

x, y - соответственно *x*- и *y*-координаты области.

Все параметры функции *brhs* являются *входными.*

Продолжение описания параметров подпрограммы FPS2H:

coefu - значение коэффициента при *u* в решаемом дифференциальном уравнении.

nx, ny - соответственно число линий сетки по осям *x* и *y*. Значения *nx* и *ny* не могут быть меньше четырех (см. также комментарий 1).

ax, bx - соответственно левая и правая *x*-координаты области интегрирования.

ay, by - соответственно нижняя и верхняя *y*-координаты области интегрирования.

ibcty - массив размера 4, означающий тип граничных условий на каждой стороне области или указывающий на периодическое решение. Стороны имеют номера (см. также рис. 4.4):

1 - правая ($x = bx$);

2 - нижняя ($y = ay$);

3 - левая ($x = ax$);

4 - верхняя ($y = by$).

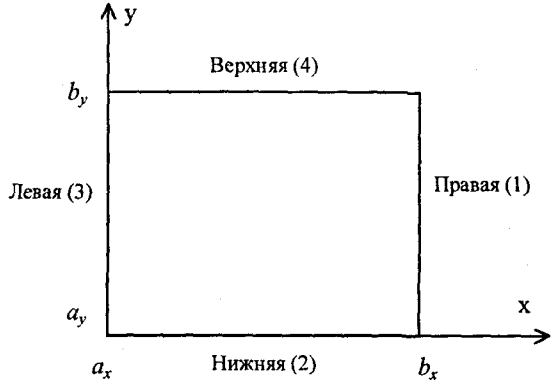


Рис. 4.4. Стороны области интегрирования

Существует 3 типа граничных условий:

- 1 - задаются значения *u* (Дирихле);
- 2 - задаются значения *du / dx* на сторонах 1 и/или 3 и *du / dy* на сторонах 2 и/или 4 (Нейман);
- периодический.

iorder - порядок точности конечно-разностной аппроксимации. Может быть равен 2 или 4. Обычно задают *iorder* = 4.

u - массив формы (*Ldu, ny*), содержащий решение в *nx*×*ny* точках сетки.

Ldu - ведущий размер массива *u*; обычно *Ldu* = *nx*.

Автоматически для решения предоставляется память:

- $(nx + 2)(ny + 2) + (nx + 1)(ny + 1)(iorder - 2)/2 + 6(nx + ny) + nx/2 + 16$ байт в случае FPS2H;
- $2(nx + 2)(ny + 2) + (nx + 1)(ny + 1)(iorder - 2) + 12(nx + ny) + nx + 32$ байт в случае DFPS2H.

Память можно выделить явно, применив F2S2H (DF2S2H):

CALL F2S2H(*prhs, brhs, coefu, nx, ny, ax, bx, ay, by, ibcty, iorder, u, Ldu, uwork, work*)

Дополнительные параметры подпрограммы F2S2H:

uwork - рабочий массив размера $(nx + 2)(ny + 2)$. Если размер массива *u* достаточно большой, то *u* и *uwork* могут быть одним массивом.

work - рабочий массив размера $(nx + 1)(ny + 1)(iorder - 2)/2 + 6(nx + ny) + nx/2 + 16$.

Комментарии:

1. Сеточные расстояния - это (в случае равномерной сетки) расстояния между линиями сетки. Вычисляются по формулам $hx = (bx - ax)/(nx - 1)$ и $hy = (by - ay)/(ny - 1)$. Сеточные расстояния по *x* и *y* должны быть одинаковыми ($hx = hy$). Чтобы ускорить быстрое преобразование Фурье, число $nx - 1$ должно быть произведением малых простых чисел, например степенью числа 2. Хорошим выбором для *nx* являются числа 17, 33 и 65.
2. Если *-coefu* почти равен собственному значению лапласиана с однородными граничными условиями, то решение может содержать большие ошибки.

Описание:

Пусть $c = coefu$, $a_x = ax$, $b_x = bx$, $a_y = ay$, $b_y = by$, $n_x = nx$ и $n_y = ny$. Подпрограмма FPS2H основана на коде HFFT2D, приведенном в [15]. Она находит решение уравнения

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + cu = p(x, y)$$

в прямоугольной области $[a_x, b_x]\times[a_y, b_y]$ с заданными пользователем начальными условиями (Дирихле, Неймана или периодическими). Стороны области нумеруются по часовой стрелке (см. рис. 4.4).

Когда $c = 0$ и заданы граничные условия Неймана или периодические, то к решению может быть добавлена любая константа. В этом случае возвращается решение, имеющее минимальную ∞-норму. Для получения решения используется конечно-разностная аппроксимация второго или четвертого порядка точности. Получаемая в результате аппроксимации линейная система алгебраических уравнений решается с применением техники быстрых преобразований Фурье. Алгоритм существенно ускоряется, если n_x является произведением небольших простых чисел. В этом случае время поиска решения пропорционально $n_x n_y \log_2 n_x$. С деталями алгоритма можно познакомиться в [15].

Если оценка $p(x, y)$ недорога, то временные различия при работе с $iorder = 2$ и $iorder = 4$ незначительны.

Пример. Решается уравнение

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+3u=-2\sin(x+2y)+16e^{2x+3y}$$

с граничными условиями $\partial u/\partial y = 2\cos(x + 2y) + 3i^{2x + 3y}$ на нижней стороне и $u = \sin(x + 2y) + e^{2x + 3y}$ на трех других сторонах. Область интегрирования - это прямоугольник [0, 1/4]×[0, 1/2]. Результатом FPS2H являются 17×33 значений массива *u*. Выполняющая квадратичную интерполяцию функция QD2VL используется для вывода на печать таблицы результатов. Также рисуется график, отображающий массив *u*.

```
program fps2h_test
 use dfimsl
 integer(4), parameter :: ncval = 11, nx = 17, nxtabl = 5, ny = 33, nytabl = 5
 integer(4) :: i, ibcty(4), iorder, j, k
 real(4) :: ax, ay, brhs, bx, by, coefu, error, prhs,                    &
       true, u(nx, ny), utabl, x, xdata(nx), y, ydata(ny)
 ! Массив, введенный для графического отображения
 ! результата - зависимости u(x, y)
 real(4), allocatable :: uxy(:, :)
 !dec$attributes array_visualizer :: uxy
 external brhs, prhs
 ! Задаем размеры прямоугольной области
 ax = 0.0; bx = 0.25
 ay = 0.0; by = 0.50
 ! Задаем тип граничных условий
 ibcty(1) = 1; ibcty(2) = 2
 ibcty(3) = 1; ibcty(4) = 1
 coefu = 3.0                          ! Коэффициент перед u
 iorder = 4                           ! Порядок точности аппроксимации
 ! Решаем дифференциальное уравнения в частных производных
 call fps2h(prhs, brhs, coefu, nx, ny, ax, bx, ay, by, ibcty, iorder, u, nx)
 ! Подготовка к квадратичной интерполяции
 do i = 1, nx
   xdata(i) = ax + (bx - ax) * float(i - 1) / float(nx - 1)
 end do
 do j = 1, ny
   ydata(j) = ay + (by - ay) * float(j - 1) / float(ny - 1)
 end do
 ! Печатаем решение и осуществляем графический вывод результата
 write(*, '(8x, a, 11x, a, 11x, a, 8x, a)') 'x', 'y', 'u', 'error'
 k = 0
```

```
 allocate(uxy(3, nxtabl * nytabl))
 do j = 1, nytabl
  y = ay + (by - ay) * float(j - 1) / float(nytabl - 1)
  do i = 1, nxtabl
    x = ax + (bx - ax) * float(i - 1) / float(nxtabl - 1)
    utabl = qd2vl(x, y, nx, xdata, ny, ydata, u, nx, .false.)
    true = sin(x + 2.0 * y) + exp(2.0 * x + 3.0 * y)
    error = true - utabl
    write(*, '(4f12.4)') x, y, utabl, error
    k = k + 1
    uxy(1, k) = x
    uxy(2, k) = y
    uxy(3, k) = utabl
  end do
 end do
 ! Используем для вывода поверхности самый темный цвет
 call vGraph2(uxy, nytabl * nytabl)     ! Результат см. на рис. 4.5
 deallocate(uxy)
end program fps2h_test

real function prhs(x, y)
 real(4) :: x, y
 ! Задаем правую часть дифференциального уравнения в частных производных
 prhs = -2.0 * sin(x + 2.0 * y) + 16.0 * exp(2.0 * x + 3.0*y)
end function prhs

real function brhs(iside, x, y)
 integer(4) :: iside
 real(4) :: x, y
 if(iside == 2) then                    ! Определяем граничные условия
   brhs = 2.0 * cos(x + 2.0 * y) + 3.0 * exp(2.0 * x + 3.0 * y)
 else
   brhs = sin(x + 2.0 * y) + exp(2.0 * x + 3.0 * y)
 end if
end function brhs

subroutine vGraph2(fun, nvalues)        ! Выводит массив как векторный граф
 use avdef                              ! отображателя массивов
 use avviewer
 use dflib
 integer(4) :: nvalues
 real(4) :: fun(3, nvalues)
 integer(4) hv, status, nError
 character(1) :: key
```

```
character(av_max_label_len) :: xLabel = 'x', yLabel = 'y', zLabel = 'u'
! Подпрограмма vGraph2 выводит в режиме векторного графа трехмерный образ,
! в то время как подпрограмма vGraph формирует плоское изображение
call faglStartWatch(fun, status)          ! Сообщаем ОМ имя отображаемого массива
print *, "Starting Array Viewer"
! Запускаем ОМ с использованием fav-подпрограммы
call favStartViewer(hv, status)
if(status /= 0) then
 call favGetErrorNo(hv, nError, status)
 if(nError /= 0) then
  print *, "Array Viewer reports error ", nError
  stop
 end if
end if
! Передаем ОМ данные подлежащего отображению массива
call favSetArray(hv, fun, status)
! Задаем заголовок экземпляра ОМ
call favSetArrayName(hv, "Van der Pol Cycle", status)
! Отображаем массив в виде векторного графа
call favSetGraphType(hv, VectorGraph, status)
! Задаем режим вывода заданных пользователем имен осей координат
call favSetUseAxisLabel(hv, x_axis, 1, status)
call favSetUseAxisLabel(hv, y_axis, 1, status)
call favSetUseAxisLabel(hv, z_axis, 1, status)
! Новые (вместо dim1, dim2 и z) имена x- и y-осей координат.
! Длина переменной, задающей имя оси, равна AV_MAX_LABEL_LEN
call favSetAxisLabel(hv, x_axis, xLabel, status)
call favSetAxisLabel(hv, y_axis, yLabel, status)
call favSetAxisLabel(hv, z_axis, zLabel, status)
! Устанавливаем режим явного задания разметок координатных осей
call favSetAxisAutoDetail(hv, 0, status)
! Число больших разметок на координатных осях
call favSetNumMajorTickmarks(hv, x_axis, 3, status)
call favSetNumMajorTickmarks(hv, y_axis, 3, status)
call favSetNumMajorTickmarks(hv, z_axis, 3, status)
! Показываем ОМ на экране
call favShowWindow(hv, av_true, status)
print *, "Press any key to close down the viewer"
key = getcharqq( )
call favEndViewer(hv, status)             ! Закрываем ОМ
call faglEndWatch(fun, status)            ! Освобождаем ресурсы
end subroutine vGraph2
```

Результат:

x	y	u	error
0.0000	0.0000	1.0000	0.0000
0.0625	0.0000	1.1956	0.0000
0.1250	0.0000	1.4087	0.0000
0.1875	0.0000	1.6414	0.0000
0.2500	0.0000	1.8961	0.0000
0.0000	0.1250	1.7024	0.0000
0.0625	0.1250	1.9562	0.0000
0.1250	0.1250	2.2345	0.0000
0.1875	0.1250	2.5407	0.0000
0.2500	0.1250	2.8783	0.0000
0.0000	0.2500	2.5964	0.0000
0.0625	0.2500	2.9322	0.0000
0.1250	0.2500	3.3034	0.0000
0.1875	0.2500	3.7148	0.0000
0.2500	0.2500	4.1720	0.0000
0.0000	0.3750	3.7619	0.0000
0.0625	0.3750	4.2163	0.0000
0.1250	0.3750	4.7226	0.0000
0.1875	0.3750	5.2878	0.0000
0.2500	0.3750	5.9199	0.0000
0.0000	0.5000	5.3232	0.0000
0.0625	0.5000	5.9520	0.0000
0.1250	0.5000	6.6569	0.0000
0.1875	0.5000	7.4483	0.0000
0.2500	0.5000	8.3380	0.0000

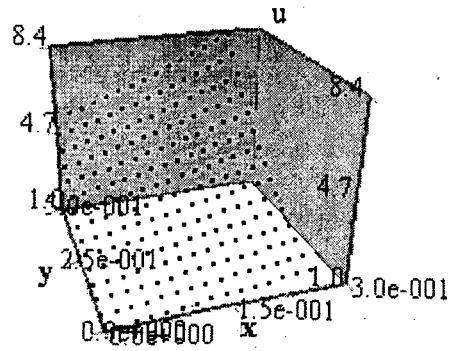


Рис. 4.5. График u(x, y)

4.6.2. ПОДПРОГРАММА FPS3H (DFPS3H)

Решает уравнения Пуассона или Гельмгольца в трехмерном прямоугольном параллелепипеде, используя быстрый пуассоновский решатель, основанный на конечно-разностной схеме на однородной сетке. Имеет вызов

CALL FPS3H(*prhs, brhs, coefu, nx, ny, nz,* &
ax, bx, ay, by, az, bz, ibcty, iorder, u, Ldu, mdu)

Параметры подпрограммы FPS3H:

Пользовательские функции: prhs, brhs.

Входные: coefu, nx, ny, nz, ax, bx, ay, by, az, bz, ibcty, iorder, Ldu, mdu.

Выходной: u.

prhs - пользовательская функция, выполняющая оценку правой части уравнения. Должна иметь атрибут EXTERNAL. Имеет вызов

result = *prhs*(*x, y, z*)

Параметры функции prhs:

x, y, z - соответственно *x*-, *y*- и *z*-координаты области; являются *входными.*

brhs - пользовательская функция, выполняющая оценку граничных условий правой части уравнения. Должна иметь атрибут EXTERNAL. Имеет вызов

resul = *brhs*(*iside, x, y, z*)

Параметры функции brhs:

iside - номер стороны; см. ниже описание параметра *ibcty*.

x, y, z - соответственно *x*-, *y*- и *z*-координаты области.

Все параметры функции *brhs* являются *входными.*

Продолжение описания параметров подпрограммы FPS2H:

coefu - значение коэффициента при *u* в решаемом дифференциальном уравнении.

nx, ny, nx - соответственно число линий сетки по осям *x, y* и *z*. Значения *nx, ny* и *nz* не могут быть меньше четырех (см. также комментарий 1).

ax, bx - соответственно левая и правая *x*-координаты области интегрирования.

ay, by - соответственно нижняя и верхняя *y*-координаты области интегрирования.

az, bz - соответственно меньшая и большая *z*-координаты области интегрирования.

ibcty - массив размера 4, означающий тип граничных условий на каждой стороне области или указывающий на периодическое решение. Стороны имеют номера (см. также рис. 4.6):

1 - правая ($x = bx$);

2 - нижняя ($y = ay$);

3 - левая ($x = ax$);

4 - верхняя ($y = by$);

5 - передняя ($z = bz$);

6 - задняя ($z = az$).

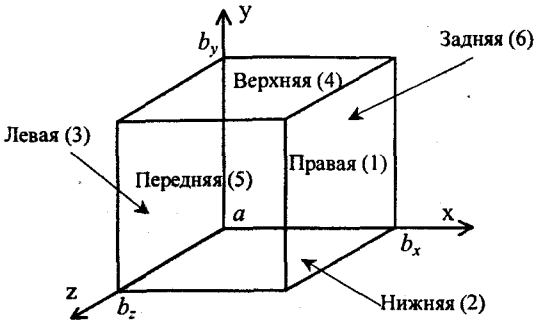


Рис. 4.6. Стороны области интегрирования

Существует 3 типа граничных условий:

- 1 - задаются значения *u* (Дирихле);
- 2 - задаются значения *du / dx* на сторонах 1 и/или 3, *du / dy* на сторонах 2 и/или 4 и *du / dz* на сторонах 5 и/или 6 (Нейман);
- периодический.

iorder - порядок точности конечно-разностной аппроксимации. Может быть равен 2 или 4. Обычно задают *iorder* = 4.

u - массив формы (*Ldu, mdu, nz*), содержащий решение в *nx*×*ny*×*nz* точках сетки.

Ldu - ведущий размер массива *u*; обычно *Ldu* = *nx*.

mdu - средний размер массива *u*; обычно *mdu* = *ny*.

Автоматически для решения предоставляется память:

- $(nx + 2)(ny + 2)(nz + 2) + (nx + 1)(ny + 1)(nz + 1)(iorder - 2)/2 + 2(nx*ny + nx*nz + ny*nz) + 2(nx + ny + 1) + \mathrm{MAX}(2nx*ny, 2nx + ny + 4nz + (nx + nz)/2 + 29)$ байт в случае FPS3H;

- $2(nx+2)(ny+2)(nz+2)+(nx+1)(ny+1)(nz+1)(iorder-2)+4(nx*ny+nx*nz+ny*nz)+4(nx+ny+1)+2\text{MAX}(2nx*ny, 2nx+ny+4nz+(nx+nz)/2+29)$ байт в случае DFPS3H;

Память можно выделить явно, применив F2S2H (DF2S2H):

CALL F2S3H(*prhs, brhs, coefu, nx, ny, nz, ax, bx, ay, by, az, bz, ibcty, iorder, u, Ldu, mdu, uwork, work*) &

Дополнительные параметры подпрограммы F2S3H:

uwork - рабочий массив размера $(nx+2)(ny+2)(nz+2)$. Если размер массива *u* достаточно большой, то *u* и *uwork* могут быть одним массивом.

work - рабочий массив размера $(nx+1)(ny+1)(nz+1)(iorder-2)/2+2(nx*ny+nx*nz+ny*nz)+2(nx+ny+1)+\text{MAX}(2nx*ny, 2nx+ny+4nz+(nx+nz)/2+29)$.

Комментарии:

1. Сеточные расстояния вычисляются по формулам $hx=(bx-ax)/(nx-1)$, $hy=(by-ay)/(ny-1)$ и $hz=(bz-az)/(nz-1)$. Сеточные расстояния по *x*, *y* и *z* должны быть одинаковыми ($hx=hy=hz$). Чтобы ускорить быстрое преобразование Фурье, числа $nx-1$ и $nz-1$ должны быть произведениями малых простых чисел. Хорошим выбором для *nx* и *nz* являются числа 17, 33 и 65.
2. Если *-coefu* почти равен собственному значению лапласиана с однородными граничными условиями, то решение может содержать большие ошибки.

Описание:

Пусть $c=coefu$, $a_x=ax$, $b_x=bx$, $a_y=ay$, $b_y=by$, $a_z=az$, $b_z=bz$, $n_x=nx$, $n_y=ny$ и $n_z=nz$. Подпрограмма FPS3H основана на коде HFFT3D, приведенном в [15]. Она находит решение уравнения

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}+cu=p(x,y,z)$$

в области $[a_x, b_x]\times[a_y, b_y]\times[a_z, b_z]$ с заданными пользователем начальными условиями (Дирихле, Неймана или периодическими). Стороны области нумеруются, как показано на рис. 4.6.

Когда $c=0$ и заданы граничные условия Неймана или периодические, то к решению может быть добавлена любая константа. В этом случае возвращается решение, имеющее минимальную ∞-норму. Для получения решения используется конечно-разностная аппроксимация второго или четвертого порядка точности. Получаемая в результате аппроксимации линейная система алгебраических уравнений решается с применением техники быстрых

преобразований Фурье. Алгоритм существенно ускоряется, если n_x - 1 и n_z - 1 являются произведениями небольших простых чисел. В этом случае время поиска решения пропорционально $n_x n_y n_z \left(\log_2^2 n_x + \log_2^2 n_z\right)$. С деталями алгоритма можно познакомиться в [15].

Если оценка $p(x, y, z)$ недорога, то временные различия при работе с *iorder* = 2 и *iorder* = 4 незначительны.

Пример. Решается уравнение

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} + 10u = -4\cos(3x + y - 2z) + 12e^{x-z} + 10$$

с граничными условиями $\partial u/\partial z = -2\sin(3x + y - 2z) - e^{x-z}$ на передней стороне и с граничными условиями $u = \cos(3x + y - 2z) + e^{x-z} + 1$ на пяти остальных сторонах области. Область интегрирования - это прямоугольный параллелепипед [0, 1/4]×[0, 1/2]×[0, 1/2]. Результатом FPS3H являются 9×17×17 значений массива *u*. Выполняющая квадратичную интерполяцию функция QD3VL используется для вывода на печать таблицы результатов.

```
program fps3h_test
 use dfimsl
 integer(4), parameter :: nx = 5, nxtabl = 4, ny = 9, nytabl = 3,                &
      nz = 9, nztabl = 3, Ldu = nx, mdu = ny
 integer(4) :: i, ibcty(6), iorder, j, k, nout
 real(4) :: ax, ay, az, brhs, bx, by, bz, coefu, prhs, u(Ldu, mdu, nz), utabl, x,  &
      error, true, xdata(nx), y, ydata(ny), z, zdata(nz)
 external brhs, prhs
 ! Задаем область интегрирования (прямоугольный параллелепипед)
 ax = 0.0; bx = 0.125
 ay = 0.0; by = 0.25
 az = 0.0; bz = 0.25
 ! Устанавливаем тип граничных условий
 ibcty(1) = 1; ibcty(2) = 1; ibcty(3) = 1
 ibcty(4) = 1; ibcty(5) = 2; ibcty(6) = 1
 coefu = 10.0                          ! Коэффициент перед u
 iorder = 4                            ! Порядок метода интегрирования
 ! Решаем исходное уравнение
 call fps3h(prhs, brhs, coefu, nx, ny, nz,                                       &
      ax, bx, ay, by, az, bz, ibcty, iorder, u, Ldu, mdu)
 do i = 1, nx                          ! Подготовка к квадратичной интерполяции
  xdata(i) = ax + (bx - ax) * float(i - 1) / float(nx - 1)
 end do
 do j = 1, ny
```

```
  ydata(j) = ay + (by - ay) * float(j - 1) / float(ny - 1)
 end do
 do k = 1, nz
  zdata(k) = az + (bz - az) * float(k - 1) / float(nz - 1)
 end do
 ! Вывод решения
 write(*, '(8x,5(a,11x))') 'x', 'y', 'z', 'u', 'error'
 do k = 1, nztabl
  z = az + (bz - az) * float(k - 1) / float(nztabl - 1)
  do j = 1, nytabl
   y = ay + (by - ay) * float(j - 1) / float(nytabl - 1)
   do i = 1, nxtabl
    x = ax + (bx - ax) * float(i - 1) / float(nxtabl - 1)
    utabl = qd3vl(x, y, z, nx, xdata, ny, ydata, nz, zdata, u, Ldu, mdu, .false.)
    true = cos(3.0 * x + y - 2.0 * z) + exp(x - z) + 1.0
    error = utabl - true
    write(*, '(5f12.4)') x, y, z, utabl, error
   end do
  end do
 end do
end program fps3h_test

real function prhs(x, y, z)
 real(4) :: x, y, z
 ! Правая часть уравнения в частных производных
 prhs = -4.0 * cos(3.0 * x + y - 2.0 * z) + 12 * exp(x - z) + 10.0
end function prhs

real function brhs(iside, x, y, z)
 integer(4) :: iside
 real(4) :: x, y, z
 ! Граничные условия
 if(iside == 5) then
  brhs = -2.0 * sin(3.0 * x + y - 2.0 * z) - exp(x - z)
 else
  brhs = cos(3.0 * x + y - 2.0 * z) + exp(x - z) + 1.0
 end if
end function brhs
```

Результат:

x	y	z	u	error
0.0000	0.0000	0.0000	3.0000	0.0000
0.0417	0.0000	0.0000	3.0348	0.0000

0.0833	0.0000	0.0000	3.0559	0.0001
0.1250	0.0000	0.0000	3.0637	0.0001
0.0000	0.1250	0.0000	2.9922	0.0000
0.0417	0.1250	0.0000	3.0115	0.0000
0.0833	0.1250	0.0000	3.0175	0.0000
0.1250	0.1250	0.0000	3.0107	0.0000
0.0000	0.2500	0.0000	2.9690	0.0001
0.0417	0.2500	0.0000	2.9731	0.0000
0.0833	0.2500	0.0000	2.9645	0.0000
0.1250	0.2500	0.0000	2.9440	-0.0001
0.0000	0.0000	0.1250	2.8514	0.0000
0.0417	0.0000	0.1250	2.9123	0.0000
0.0833	0.0000	0.1250	2.9592	0.0000
0.1250	0.0000	0.1250	2.9922	0.0000
0.0000	0.1250	0.1250	2.8747	0.0000
0.0417	0.1250	0.1250	2.9211	0.0010
0.0833	0.1250	0.1250	2.9524	0.0010
0.1250	0.1250	0.1250	2.9689	0.0000
0.0000	0.2500	0.1250	2.8825	0.0000
0.0417	0.2500	0.1250	2.9123	0.0000
0.0833	0.2500	0.1250	2.9281	0.0000
0.1250	0.2500	0.1250	2.9305	0.0000
0.0000	0.0000	0.2500	2.6314	-0.0249
0.0417	0.0000	0.2500	2.7420	-0.0004
0.0833	0.0000	0.2500	2.8112	-0.0042
0.1250	0.0000	0.2500	2.8609	-0.0138
0.0000	0.1250	0.2500	2.7093	0.0000
0.0417	0.1250	0.2500	2.8153	0.0344
0.0833	0.1250	0.2500	2.8628	0.0242
0.1250	0.1250	0.2500	2.8825	0.0000
0.0000	0.2500	0.2500	2.7351	-0.0127
0.0417	0.2500	0.2500	2.8030	-0.0011
0.0833	0.2500	0.2500	2.8424	-0.0040
0.1250	0.2500	0.2500	2.8735	-0.0012

4.7. ЗАДАЧА ШТУРМА - ЛИУВИЛЛЯ

4.7.1. ПОДПРОГРАММА SLEIG (DSLEIG)

Определяет собственные значения, собственные функции и/или спектральную функцию задачи Штурма - Лиувилля

$$-\frac{d}{dx}\left(p(x)\frac{du}{dx}\right)+q(x)u=\lambda r(x)u,\ x\in(a,b) \tag{4.12}$$

с граничными условиями (в регулярных точках)

$$\begin{aligned} &a_1u(a)-a_2(p(a)u'(a))=\lambda(a_1'u(a)-a_2'(p(a)u'(a)));\\ &b_1u(b)+b_2(p(b)u'(b))=0. \end{aligned} \tag{4.13}$$

Напомним понятия собственного значения, собственной функции и спектральной функции задачи (4.12).

Число λ_0 называется *собственным значением задачи* (4.12), если при $\lambda=\lambda_0$ уравнение (4.12) имеет нетривиальное решение $y_0(x)$, не равное тождественно нулю, удовлетворяющее граничным условиям (4.13); при этом функция $y_0(x)$ называется *собственной функцией*, отвечающей собственному значению λ_0.

Рассмотрим теперь начальную задачу, в которой дифференциальное уравнение (4.12) исследуется на полуоси $0\le x<\infty$ с начальным условием

$$y'(0)-hy(0)=0. \tag{4.14}$$

Пусть $\varphi(x,\lambda)$ - решение уравнения (4.12) с начальными условиями $y(0)=1$ и $y'(0)=h$, т. е. $\varphi(x,\lambda)$ удовлетворяет начальному условию (4.14). Пусть $f(x)$ - любая функция из $L_{2,\rho}(0,\infty)$ и

$$\Phi_{f,b}(x)=\int_0^b f(x)\varphi(x,\lambda)\,dx,$$

где b - произвольное конечное положительное число. Тогда для каждой действительной функции $q(x)$, суммируемой в каждом конечном подынтервале интервала $[0,\infty)$, и каждого действительного числа h есть по крайней мере одна не зависящая от $f(x)$ неубывающая функция $\rho(\lambda)$, $-\infty<\lambda<\infty$, обладающая следующими свойствами:

1) существует функция $\Phi_f(\lambda)$, являющаяся пределом $\Phi_{f,\,b}(\lambda)$ при $b\to\infty$ в метрике $L_{2,\rho}(-\infty,\infty)$, т. е.

$$\lim_{b\to\infty}\int_{-\infty}^{\infty}\left|\Phi_f(\lambda)-\Phi_{f,b}(\lambda)\right|^2 d\rho(\lambda)=0;$$

2) имеет место неравенство Парсеваля

$$\int_0^\infty |f(x)|^2 dx = \int_{-\infty}^{\infty} |\Phi_f(\lambda)|^2 d\rho(\lambda) = 0 .$$

Функция $\rho(\lambda)$ называется *спектральной функцией* (или *спектральной плотностью*) начальной задачи (4.12), (4.14).

Подпрограмма SLEIG имеет вызов

CALL SLEIG(*cons, coeffn, endfin, numeig, index, tevlab, tevlrl, eval*)

Параметры подпрограммы SLEIG:

Входные: cons, coeffn, endfin, numeig, index, tevlab, tevlrl.

Выходной: eval.

cons - вектор из восьми элементов, содержащий в *cons*(1), ..., *cons*(8) соответственно a_1, a'_1, a_2, a'_2, b_1, b_2, a, b.

coeffn - пользовательская подпрограмма, оценивающая коэффициенты функций. Имеет вызов

CALL *coeffn*(*x, px, qx, rx*)

Параметры подпрограммы coeffn:

x - независимая переменная.

px, qx, rx - соответственно значения $p(x)$, $q(x)$, $r(x)$ в x.

Параметр *x* является *входным*, параметры *px, qx* и *rx* - *выходные*.

Имя *coeffn* должно в вызывающей программной единице получить атрибут EXTERNAL.

Продолжение описания параметров подпрограммы SLEIG:

endfin - логический массив размера 2; *endfin*(1) = .TRUE., если граничная точка *a* является конечной; *endfin*(2) = .TRUE., если конечна граничная точка *b*.

numeig - число подлежащих определению собственных значений.

index - вектор размера *numeig*, содержащий индексы подлежащих определению собственных значений.

tevlab - допуск абсолютной ошибки для собственных значений.

telvrl - допуск относительной ошибки для собственных значений.

eval - массив размера *numeig*, содержащий вычисленные приближения собственных значений, индексы которых заданы в векторе *index*.

Автоматически для решения предоставляется память:

- 4*numeig* + MAX(1000, *numeig* + 22) байт в случае S2EIG;
- 8*numeig* + MAX(1000, *numeig* + 22) байт в случае DS2EIG.

Память можно выделить явно, применив S2EIG (DS2EIG):

CALL S2EIG(*const, coeffn, endfin, numeig, index, tevlab, tevlrl, eval, job, iprint, tols, numx, xef, nrho, t, type, ef, pdef, rho, iflag, work, iwork*)

Дополнительные параметры подпрограммы S2EIG:

Входные: job, iprint, tols, xef, nrho, t.

Входные/выходные: numx, type.

Выходные: ef, pdef, rho, iflag.

Рабочие массивы: work, iwork.

job - логический массив размера 5. Имеет следующий смысл:

- *job*(1) = .TRUE., если вычисляются собственные значения, а не соответствующие им собственные функции;
- *job*(2) = .TRUE., если вычисляются и собственные значения, и соответствующие им собственные функции;
- *job*(3) = .TRUE., если вычисляется спектральная функция на некотором подынтервале;
- *job*(4) = .TRUE., если отменяется нормальная автоматическая классификация. Если *job*(4) = .TRUE., то массив *type* должен быть определен правильно. Большинство пользователей не желают применять отмену, но она может быть уместной при исследовании задач, в которых функция коэффициентов не ведет себя как степенная вблизи вырожденных граничных точек. Классификация важна при вычислении спектральной функции, поэтому параметр *job*(4) игнорируется, когда *job*(3) = .TRUE.;
- *job*(5) = .TRUE., если сетка, на которой выполняются вычисления, создается подпрограммой SLEIG. Если *job*(5) = .TRUE. и *numx* = 0, то число точек сетки также выбирается SLEIG. Если *numx* > 0, то сетка будет содержать *numx* точек. Если *job*(5) = .FALSE., то *numx* и вектор *xef*(*) должны быть заданы пользователем.

iprint - параметр, управляющий внутренней печатью. Если *iprint* = 0, то печать не выполняется. Если *job*(1) или *job*(2) истинны, *iprint* принимает следующие значения:

- 1 - выводится начальная сетка (первые 51 или меньше точек), собственные значения оцениваются на каждом уровне;
- 4 - то же, что и при *iprint* = 1, плюс на каждом уровне значения *x*(*), *ef*(*) и *pdef*(*);
- 5 - все, что для 1 и 4, плюс на каждом уровне границы поиска собственного значения, промежуточные данные о собственной функции и ее норме.

Если *job*(3) = .TRUE., *iprint* принимает следующие значения:

- 1 - реальный использованный на каждой итерации интервал (*a*, *b*) и общее число вычисленных собственных значений;
- 2 - то же, что и при *iprint* = 1, плюс асимптоты асимптотических формул и промежуточные значения $\rho(t)$;
- 3 - все, что для 1 и 2, плюс начальная сетка на каждой итерации, индекс наибольшего собственного значения, которое может быть вычислено, и различные собственные значения и значения R_n;
- 4 - все, что для 1-3, плюс значения $\hat{\rho}$ на каждом уровне;
- 5 - все, что для 1-4, плюс R_n и собственные значения, лежащие ниже асимптоты.

Если *job*(4)=.FALSE., *iprint* принимает следующие значения:

- 2 - выходное описание спектра;
- 3 - то же, что и при *iprint* = 2, плюс константы для граничных условий Фридрихса;
- 5 все, что для 1 и 2, плюс промежуточная информация о расчетах классификации.

tols - вектор размера 4, содержащий следующие допуски:

- *tols*(1) - допуск абсолютной ошибки для собственных функций;
- *tols*(2) - допуск относительной ошибки для собственных функций;
- *tols*(3) - допуск абсолютной ошибки для производных собственных функций;
- *tols*(4) - допуск относительной ошибки для производных собственных функций.

Все допуски положительны; относительный допуск не может быть меньше 100*AMACH(4).

numx - число выходных точек (размер вектора *xef*), в которых оценивается каждая собственная функция, когда *job*(2) = .TRUE.. Если *job*(5)= .FALSE. и *numx* > 0, *numx* задает число точек в начальной сетке. Поскольку граничные точки *a* и *b* должны принадлежать сетке, *numx* не может в этом случае равняться единице. Если *job*(5) = .FALSE. и *job*(3) = .TRUE., то число *numx* должно быть больше нуля. На выходе, когда на входе *numx* = 0 и *job*(2) или *job*(5) = .TRUE., *numx* содержит число точек, в которых выполнялась оценка собственных функций.

xef - вектор, содержащий, если *job*(2) = .TRUE., точки, в которых желательно выполнить оценку собственной функции. В противном случае, если *job*(5) = .FALSE. и *numx* > 0, в вектор *xef* вводится задаваемая пользователем

начальная сетка. Элементы вектора упорядочены так, что $a = xef(1) < xef(2) < \ldots < xef(numx) = b$. Если какая-либо граница является бесконечностью, соответствующий элемент вектора - *xef*(1) или *xef*(*numx*) - игнорируется. Однако необходимо, чтобы элемент *xef*(2) был отрицателен, когда *endfin*(1)= = .FALSE., и чтобы *xef*(*numx* - 1) был положителен, когда *endfin*(2) = = .FALSE.. На выходе *xef* изменяется, лишь когда *job*(2) и *job*(5) истинны. Если *job*(2) = .FALSE., вектор *xef* не адресуется. Если *job*(2) = .TRUE. и *numx* > 0, размер вектора *xef* должен быть не менее *numx* + 16. Если *job*(2) = .TRUE. и *numx* = 0, размер вектора *xef* должен быть не менее 31.

nrho - число значений, которые надо получить на выходе в векторе *rho*(*). Параметр *nrho* не используется, если *job*(3) = .FALSE..

t - вещественный вектор размера *nrho*, содержащий подынтервалы, на которых следует определять спектральные функции *rho*. Элементы вектора должны задаваться в возрастающем порядке. Существование и размещение непрерывного спектра может быть определено посредством вызова SLEIG, в котором первые 4 элемента *job* равны .FALSE., а *iprint* = 1. Вектор *t* не используется, когда *job*(3) = .FALSE..

type - логическая матрица формы (4, 2). Столбец 1 содержит данные о граничной точке *a*, столбец 2 - о граничной точке *b*. При этом

- *type*(1, :) = .TRUE., если только одна точка является регулярной;
- *type*(2, :) = .TRUE., если граничная точка является предельным кругом;
- *type*(3, :) = .TRUE., если граничная точка не является осциллятором для всех собственных значений;
- *type*(4, :) = .TRUE., если граничная точка является осциллятором для всех собственных значений.

В противном случае массив *type* содержит выходные данные.

Замечание. Все эти значения должны быть заданы корректно, если *job*(4) = .TRUE..

ef - вектор значений собственных функций. Причем *ef*((*k* - 1) * *numx* + *i*) содержит оценку *u*(*xef*(*i*)), соответствующую собственному значению в *ev*(*k*). Если *job*(2) = .FALSE., то вектор *ef* не адресуется. Если *job*(2) = .TRUE. и *numx* > 0, то размер *ef* не должен быть менее *numx* * *numeig*. Если *job*(2) = .TRUE. и *numx* = 0, то размер *ef* должен быть равен 31 * *numeig*.

pdef - вектор, содержащий значения производных собственных функций. Причем *pdef*((*k* - 1) * *numx* + *i*) содержит оценку $p(xef(i))u'(xef(i))$, отвечающую собственному числу *ev*(*k*). Если *job*(2) = .FALSE., то вектор *pdef* не адресуется. Если *job*(2) = .TRUE., то размер *pdef* должен совпадать с размером вектора *ef*.

rho - вектор размера *nrho*, содержащий значения спектральной функции $\rho(t)$, $rho(i) = \rho(t(i))$. Вектор *rho* не адресуется, если *job*(3) = .FALSE..

iflag - вектор размера MAX(1, *numeig*), содержащий данные о результате. Когда *job*(1) или *job*(2) = .TRUE., значение *iflag*(*k*) относится к собственному значению с номером *k*, в противном случае употребляется только *iflag*(1). Отрицательные значения *iflag* связаны с фатальными ошибками, при возникновении которых вычисления прекращаются. Положительные величины указывают на предупреждения. Вектор *iflag* принимает следующие значения:

- -1 - необходимо слишком большое число уровней для определения собственного значения. Задача чрезмерно сложна при заданном допуске. Возможно, функции коэффициентов не являются гладкими;
- -2 - необходимо слишком большое число уровней для нахождения собственной функции. Задача чрезмерно сложна при заданном допуске. Возможно, собственные функции являются плохо обусловленными;
- -3 - необходимо слишком большое число уровней для нахождения спектральной функции. Задача чрезмерно сложна при заданном допуске;
- -4 - пользователь пытается определить спектральную функцию в задаче, не имеющей непрерывного спектра;
- -5 - пользователь пытается определить спектральную функцию в задаче, обе граничные точки которой генерируют существенный спектр;
- -6 - пользователь пытается найти спектральную функцию в задаче, относящейся ко 2-й спектральной категории, в которой надлежащая нормализация решения в граничных точках неизвестна. Например, задача с нерегулярной вырожденной или бесконечной граничной точкой на одном конце интервала и с непрерывным спектром на другом;
- -7 - проблемы в определении границ поиска;
- -8 - слишком маленький шаг интегрирования. Возможно, значения вектора *tols* слишком малы для решаемой задачи;
- -9 - слишком маленький шаг оценки спектральной функции, для которой в конечной граничной точке генерируется непрерывный спектр;
- -10 - слишком большой аргумент для круговой функции. Следует попытаться решить задачу на другой сетке или, если задача вырожденная, уменьшить интервал поиска;
- -15 - функции $p(x)$ и $r(x)$ неположительны на интервале (a, b);

- -20 - собственные значения и/или собственные функции ищутся в вырожденной задаче с осциллятором в граничной точке. Следует уменьшить интервал поиска;
- 1 - неудача, возникшая, возможно, из-за наличия собственных значений, которые не могут быть разделены подпрограммой. Вычисления продолжаются, но полученные собственные функции могут быть неточными. Следует попытаться решить задачу, задав меньшие значения допусков;
- 2 - существует неопределенность классификации при заданном допуске;
- 3 - возможно, некоторые собственные значения входят в существенный спектр. Используйте значение *iprint* большее нуля, чтобы получить дополнительную информацию о положении собственных значений в задаче со ступенчатой функцией, с тем чтобы затем получить оценки реальных собственных значений, входящих в существенный спектр;
- 4 - была выполнена замена переменных, чтобы избежать возможной медленной сходимости. Однако оценка глобальной ошибки может быть ненадежной. Рекомендуется провести вычисления с различными допусками;
- 6 - имеются проблемы со сходимостью процесса поиска собственных функций при вычислении спектральной функции. Результат $\rho(t)$ может быть неточным.

work - вещественный рабочий массив размера MAX(1000, *numeig* + 22).

iwork - целочисленный рабочий массив размера *numeig* + 3.

Описание:

Подпрограмма SLEIG решает задачу Штурма - Лиувилля в форме (4.12) с граничными условиями (4.13) (в регулярных точках), вычисляя собственные значения, отвечающие им собственные функции и/или спектральные функции. Причем

$$a_1' a_2 - a_1 a_2' > 0,$$

когда $a_1' \neq 0$ и $a_2' \neq 0$.

Задача Штурма - Лиувилля занимает видное место в спектральной теории операторов. Также она связана с некоторыми видами уравнений математической физики. Например, задача о колебаниях однородной струны, закрепленной на концах, приводит к задаче Штурма - Лиувилля для уравнения $-y'' = \lambda y$ с граничными условиями $y(0) = y(\pi) = 0$.

Задача считается *регулярной*, если

- a и b конечны;
- $p(x)$ и $r(x)$ положительны на интервале (a, b);
- $1/p(x)$, $q(x)$ и $r(x)$ локально интегрируемы около конечных точек.

В противном случае задача называется *вырожденной*. Теория предполагает, что p, p', q и r непрерывны на интервале (a, b), однако конечное число разрывов может быть обработано подпрограммой при надлежащем задании входной сетки.

В случае регулярной задачи существует бесконечное число собственных значений

$$\lambda_0 < \lambda_1 < \ldots < \lambda_k,\ k \to \infty.$$

Каждому из них отвечает с точностью до константы собственная функция.

В вырожденных задачах для определения собственных значений используются различные подходы. Так, в [46] уравнение (4.12) заменяется задачей

$$-\left(\hat{p}\hat{u}'\right)' + \hat{q}\hat{u} = \hat{\lambda}\hat{r}\hat{u} \qquad (4.15)$$

с граничными условиями

$$a_1\hat{u}(a) - a_2(\hat{p}(a)\hat{u}'(a)) = \hat{\lambda}(a_1'\hat{u}(a) - a_2'(\hat{p}(a)\hat{u}'(a)));$$
$$b_1\hat{u}(b) + b_2(\hat{p}(b)\hat{u}'(b)) = 0,$$

где $\hat{p}, \hat{q}, \hat{r}$ - ступенчатые функции, аппроксимирующие соответственно p, q и r. На сетке $a = x_1 < x_2 < \ldots < x_{n+1} = b$ ступенчатые функции обычно строятся по средней точке интерполяции. Например:

$$\hat{p}(x) = p_i \equiv p\left(\frac{x_i + x_{i+1}}{2}\right)$$

для $x \in (x_i, x_{i+1})$. Аналогично определяются аппроксимации функций коэффициентов $\hat{q}$ и $\hat{r}$. Такой подход хорошо работает в регулярных задачах. В некоторых вырожденных задачах необходимо применять более точные аппроксимации, чтобы отследить асимптотическое поведение функций. В случае ступенчатой интерполяции по средней точке решение дифференциального уравнения (4.15) на интервале (x_i, x_{i+1}) аппроксимируется схемой

$$\hat{u}(x) = \hat{u}(x_i)\phi_i'(x - x_i) + (\hat{p}\hat{u}')(x_i)\phi_i(x - x_i)/p_i$$

с

$$\phi_i(t) = \begin{cases} (\sin\omega_i t)/\omega_i, & \tau_i > 0; \\ (\sinh\omega_i t)/\omega_i, & \tau_i < 0; \\ t, & \tau_n = 0, \end{cases}$$

где

$$\tau_i = (\hat{\lambda}r_i - q_i)/p_i$$

и

$$\omega_i = \sqrt{|\tau_i|}\,.$$

Начиная с $\hat{u}(a)$ и $p'(a)\hat{u}'(a)$, совместимых с граничными условиями

$$\hat{u}(a) = a_2 - a_2'\hat{\lambda};$$

$$\hat{p}(a)\hat{u}'(a) = a_1 - a_1'\hat{\lambda},$$

вычисляются для $i = 1, 2, ..., n$ значения

$$\hat{u}(x_{i+1}) = \hat{u}(x_i)\phi_i'(h_i) + \hat{p}(x_i)\hat{u}'(x_i)\phi_i(h_i)/p_i;$$

$$\hat{p}(x_{i+1})\hat{u}'(x_{i+1}) = -\tau_i p_i \hat{u}(x_i)\phi_i'(h_i) + \hat{p}(x_i)\hat{u}'(x_i)\phi_i(h_i),$$

т. е. реализуется метод пристрелки.

При фиксированной сетке можно продолжать итерации по уточнению собственного значения до тех пор, пока не удовлетворены граничные условия в точке b. Ошибка аппроксимации λ_k составляет $O(h^2)$.

Спектральная функция задачи (4.15)

$$\hat{\rho}(t) = \sum \frac{1}{\int \hat{r}(x)\hat{u}_k^2(x)dx + \alpha},$$

где сумма определяется для k, при которых $\hat{\lambda}_k \le t$ и $\alpha = a_1'a_2 - a_1a_2' > 0$.

Пример 1. Вычисляются первые 10 собственных значений приведенной в [53] задачи:

$p(x) = r(x) = 1;$
$q(x) = x;$
$(a, b) = [0, \infty);$
$u(a) = u(b) = 0.$

Ее собственные значения известны и равны корням функции

$$f(\lambda) = J_{1/3}\left(\frac{2}{3}\lambda^{3/2}\right) + J_{-1/3}\left(\frac{2}{3}\lambda^{3/2}\right).$$

Для каждого собственного значения λ_k программа *sleig*1 выводит k, λ_k и $f(\lambda_k)$.

```
program sleig1
 use dfimsl
integer(4) :: i, index(10), numeig
real(4) :: cons(8), eval(10), lambda, tevlab, tevlrl, xnu
complex(4) :: cbs1(1), cbs2(1), z
logical(4) :: endfin(2)
external coeff
! Задаем граничные условия
```

```
cons(1) = 1.0; cons(2) = 0.0; cons(3) = 0.0; cons(4) = 0.0
cons(5) = 1.0; cons(6) = 0.0; cons(7) = 0.0; cons(8) = 0.0
endfin(1) = .true.; endfin(2) = .false.
! Будем вычислять первые 10 собственных значений
numeig = 10
do i = 1, numeig
  index(i) = i - 1
end do
! Задаем абсолютный и относительный допуски
tevlab = 10.0 * amach(4)
tevlrl = sqrt(amach(4))
call sleig(cons, coeff, endfin, numeig, index, tevlab, tevlrl, eval)
xnu = -1.0 / 3.0
write(*, "(/, 2x, 'index', 5x,'lambda', 5x, 'f(lambda)', /)")
do i = 1, numeig
  lambda = eval(i)
  z = cmplx(2.0 / 3.0 * lambda * sqrt(lambda), 0.0)
  call cbjs(xnu, z, 1, cbs1)
  call cbjs(-xnu, z, 1, cbs2)
  write(*, "(i5, f13.4, e15.4)") i - 1, lambda, real(cbs1(1) + cbs2(1))
end do
end program sleig1

subroutine coeff(x, px, qx, rx)
real(4) :: x, px, qx, rx
px = 1.0
qx = x
rx = 1.0
end subroutine coeff
```

Результат:

index	lambda	f(lambda)
0	2.3381	0.4441E-04
1	4.0879	-0.1679E-04
2	5.5205	0.6716E-04
3	6.7867	-0.4001E-05
4	7.9440	0.9022E-04
5	9.0227	0.1121E-04
6	10.0401	0.1051E-03
7	11.0084	-0.8147E-04
8	11.9361	-0.2550E-04
9	12.8293	0.2316E-03

Пример 2. В приведенной в [48] задаче

$$p(x) = r(x) = 1;$$
$$q(x) = x^2 + x^4;$$
$$(a, b) = (-\infty, \infty);$$
$$u(a) = u(b) = 0.$$

Программа находит первое собственное значение и отвечающую ему собственную функцию. По значениям невязок

$$-\frac{d}{dx}\left(p(x)\frac{du}{dx}\right) + q(x)u - \lambda r(x)u$$

выполняется грубая оценка правильности результата. Для оценки значения $(p(x)u')'$ производная u' приближается интерполяционным сплайном, возвращаемым функцией CSDER.

```
program sleig2
 integer(4) :: i, iflag(1), index(1), iwork(100), nintv, nout, nrho, numeig, numx
 real(4) :: brkup(61), cons(8), cscfup(4, 61), ef(61), eval(1),                    &
       lambda, pdef(61), px, qx, residual, rho(1), rx, t(1),                       &
       tevlab, tevlrl, tols(4), work(3000), x, xef(61)
 logical(4) :: endfin(2), job(5), type(4, 2)
 ! Массивы, введенные для графического отображения
 ! результата - зависимости u(x) и u'(x)
 real(4), allocatable :: ux(:, :), up(:, :)
 !dec$attributes array_visualizer :: ux
 external coeff
 ! Задаем граничные условия
 cons(1) = 1.0; cons(2) = 0.0; cons(3) = 0.0; cons(4) = 0.0
 cons(5) = 1.0; cons(6) = 0.0; cons(7) = 0.0; cons(8) = 0.0
 ! Будем вычислять собственное значение и собственную функцию
 job(1) = .false.; job(2) = .true.; job(3) = .false.
 job(4) = .false.; job(5) = .false.
 endfin(1) = .false.
 endfin(2) = .false.
 ! Будем вычислять собственное значение с индексом 0
 numeig = 1; index(1) = 0
 tevlab = 1.0e-3; tevlrl = 1.0e-3
 tols(1) = tevlab; tols(2) = tevlrl
 tols(3) = tevlab; tols(4) = tevlrl
 nrho = 0
 ! Задаем сетку и точки, в которых вычисляются u и u'
 numx = 61
```

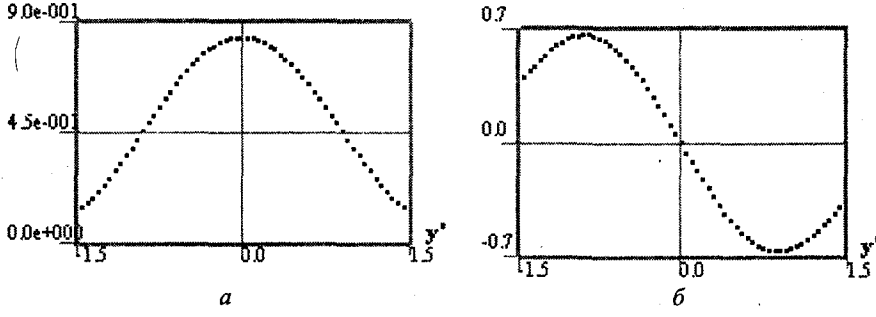
```
do i = 1, numx
  xef(i) = 0.05 * real(i - 31)
end do
call s2eig(cons, coeff, endfin, numeig, index, tevlab, tevlrl, eval,                &
      job, 0, tols, numx, xef, nrho, t, type, ef, pdef, rho, iflag, work, iwork)
lambda = eval(1)
! Аппроксимируем u' посредством интерполяционного кубического сплайна Акимы
call csakm(numx, xef, pdef, brkup, cscfup)
nintv = numx - 1
write(*, "(/, a14, f10.5, /)") ' lambda = ', lambda
write(*, "(7x, 'x', 11x, 'u(x)', 10x, 'u''(x)', 9x, 'residual', /)")
! Вычисляем невязки для подмножества точек входной сетки
! residual = ABS(-(u')' + q(x)u - lambda * u); известно, что p(x) = 1 и r(x) = 1
do i = 1, 41, 2
  x = xef(i + 10)
  call coeff(x, px, qx, rx)
  ! Функция CSDER оценивает производную кубического сплайна
  residual = abs(-csder(1, x, nintv, brkup, cscfup) + qx * ef(i + 10) - lambda * ef(i + 10))
  write (*, "(5x, f4.1, 3f15.5)") x, ef(i + 10), pdef(i + 10), residual
end do
allocate(ux(2, numx), up(2, numx))    ! Графический вывод результата -
ux(1, :) = xef; up(1, :) = xef        ! зависимостей u(x) и u'(x)
ux(2, :) = ef; up(2, :) = pdef
! Текст подпрограммы vGraph приведен в примере 1,
! сопровождающем описание подпрограммы DASPG (разд. 4.3)
call vGraph(ux, numx)                 ! Результат см. на рис. 4.7, а
call vGraph(up, numx)                 ! Результат см. на рис. 4.7, б
deallocate(ux, up)
end program sleig2

subroutine coeff(x, px, qx, rx)
 real(4) :: x, px, qx, rx
 px = 1.0
 qx = x * x + x * x * x * x
 rx = 1.0
end subroutine coeff
```

Результат:

```
lambda = 1.39247
```

x	u(x)	u'(x)	residual
-1.0	0.38632	0.65019	0.00189
-0.9	0.45218	0.66372	0.00081
-0.8	0.51837	0.65653	0.00023

-0.7	0.58278	0.62827	0.00113
-0.6	0.64334	0.57977	0.00183
-0.5	0.69812	0.51283	0.00230
-0.4	0.74537	0.42990	0.00273
-0.3	0.78366	0.33393	0.00265
-0.2	0.81183	0.22811	0.00273
-0.1	0.82906	0.11570	0.00278
0.0	0.83473	0.00000	0.00136
0.1	0.82893	-0.11568	0.00273
0.2	0.81170	-0.22807	0.00273
0.3	0.78353	-0.33388	0.00267
0.4	0.74525	-0.42983	0.00265
0.5	0.69800	-0.51274	0.00230
0.6	0.64324	-0.57967	0.00182
0.7	0.58269	-0.62816	0.00113
0.8	0.51828	-0.65641	0.00023
0.9	0.45211	-0.66361	0.00081
1.0	0.38626	-0.65008	0.00189



а *б*

Рис. 4.7. Графики функций: а - u(x); б - u'(x)

4.7.2. ПОДПРОГРАММА SLCNT (DSLCNT)

Вычисляет на указанном принадлежащем (*a*, *b*) отрезке [α, β] индексы (номера) собственных значений задачи Штурма - Лиувилля, заданной уравнением (4.12) с граничными условиями (4.13) (в регулярных точках). Имеет вызов

CALL SLCNT(*alpha, beta, cons, coeffn, endfin, ifirst, ntotal*)

Параметры подпрограммы SLCNT:

Входные: alpha, beta, cons, coeffn, endfin.

Выходной: ifirst, ntotal.

alpha, beta - соответственно левая и правая границы отрезка поиска;

cons - вектор из восьми элементов, содержащий в *cons*(1), ..., *cons*(8) соответственно a_1, a'_1, a_2, a'_2, b_1, b_2, a, b.

coeffn - пользовательская подпрограмма, оценивающая коэффициенты функций. Имеет вызов

CALL *coeffn*(*x, px, qx, rx*)

Параметры подпрограммы coeffn:

Входной: x.

Выходные: px, qx, rx.

x - независимая переменная.

px, qx, rx - соответственно значения $p(x)$, $q(x)$, $r(x)$ в x.

Продолжение описания параметров подпрограммы SLCNT:

Имя *coeffn* должно в вызывающей программной единице получить атрибут EXTERNAL.

endfin - логический массив размера 2; *endfin*(1) = .TRUE., если граничная точка *a* является конечной; *endfin*(2) = .TRUE., если конечна граничная точка *b.*

ifirst - индекс первого собственного значения, большего или равного α.

ntotal - общее число собственных значений на заданном отрезке $[\alpha, \beta]$.

Описание:

Подпрограмма SLCNT основана на процедуре INTERV из пакета SLEDGE и предназначена для совместного употребления с подпрограммой SLEIG.

Пример. Рассмотрим гармонический осциллятор [53]

$$p(x) = 1,\ q(x) = x^2,\ r(x) = 1;$$

$$(a, b) = (-\infty, \infty);$$

$$u(a) = 0,\ u(b) = 0.$$

Собственные значения задачи известны:

$$\lambda_k = 2k + 1,\ k = 0, 1, \ldots .$$

Следовательно, на интервале (10, 16) подпрограмм SLCNT должна найти 3 собственных значения, первое из которых должно иметь индекс 5.

```
program slcntTest
 integer(4) :: ifirst, nout, ntotal
 real(4) :: alpha, beta, cons(8)
 logical(4) :: endfin(2)
 external coeffn                    ! Не забываем объявить coeffn как EXTERNAL
 ! Задаем u(a) = 0, u(b) = 0
 cons(1) = 1.0e0; cons(2) = 0.0e0
 cons(3) = 0.0e0; cons(4) = 0.0e0
 cons(5) = 1.0e0; cons(6) = 0.0e0
 cons(7) = 0.0e0; cons(8) = 0.0e0
 endfin(1) = .false.
 endfin(2) = .false.
 alpha = 10.0
 beta = 16.0
 call slcnt(alpha, beta, cons, coeffn, endfin, ifirst, ntotal)
 write(*, "(/, 'Index of first eigenvalue in (', f5.2, ',', f5.2, ') is ', i2)") alpha, beta, ifirst
 write(*, "('Total number of eigenvalues in this interval: ', i2)") ntotal
end program slcntTest

subroutine coeffn(x, px, qx, rx)
 real(4) :: x, px, qx, rx
 px = 1.0e0
 qx = x * x
 rx = 1.0e0
end subroutine coeffn
```

Результат:

```
Index of first eigenvalue in (10.00, 16.00) is 5
Total number of eigenvalues in this interval: 3
```

5. РЕШЕНИЕ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ В ЧАСТНЫХ ПРОИЗВОДНЫХ

5.1. ПОДПРОГРАММА PDE_1D_MG БИБЛИОТЕКИ IMSL 90 MP

Решает уравнения в частных производных вида

$$u_t \equiv \frac{\partial u}{\partial t} = f(u, x, t), x_L < x < x_R, t > t_0 .$$

Подпрограмма требует задания начальных и граничных условий для u_t. В подпрограмме PDE_1D_MG применяется эффективный выбор зависящей от времени сетки интегрирования по пространственной переменной x. С деталями выбора сетки можно ознакомиться в [14].

Класс задач, решаемых PDE_1D_MG, описывается системой

$$\sum_{k=1}^{npde} C_{j,k}(x,\ t,\ u,\ u_x)\frac{\partial u^k}{\partial t} =$$

$$= x^{-m}\frac{\partial}{\partial x}\left(x^m R_j(x,\ t,\ u,\ u_x)\right) - Q_j(x;\ t,\ u,\ u_x); \qquad (5.1)$$

$$j = 1, \ldots, npde,\ x_L < x < x_R,\ t > t_0,\ m \in \{0,\ 1,\ 2\},$$

в которой $npde \geq 1$ - число дифференциальных уравнений. Функции R_j и Q_j могут в специальных случаях рассматриваться как поток и источник. Функции u, C_{jk}, R_j и Q_j являются непрерывными. Допустимые значения $m = 0$, $m = 1$ и $m = 2$ предназначены соответственно для задач в декартовой (картезианской), цилиндрической и сферической системах координат. В случаях, когда $m > 0$, интервал $x_L < x < x_R$ не должен содержать $x = 0$ в качестве внутренней точки. Решением системы (5.1) является вектор $u = (u^1, \ldots, u^{npde})^T$.

Граничные условия имеют вид

$$\beta_j(x,t)R_j(x,t,u,u_x) = \gamma_j(x,t,u,u_x) \qquad (5.2)$$

при $x = x_L$ и $x = x_R, j = 1, \ldots, npde$.

В уравнении граничных условий (5.2) β_j и γ_j являются непрерывными функциями. В случаях, когда $m > 0$ и граничная точка находится в нуле, конечное решение в $x = 0$ должно быть гарантировано. Это требует либо спецификации решения при $x = 0$, либо означает, что

$$R_j\big|_{x=x_L} = 0 \text{ или } R_j\big|_{x=x_R} = 0.$$

Начальные значения удовлетворяют равенству

$$u(x, t_0) = u_0(x), x \in (x_L, x_R),$$

где u_0 - кусочно-непрерывный вектор-функция от x с *npde* компонентами.

Функции C_{jk}, R_j, Q_j, β_j, γ_j и u_0 известны, задаются пользователем и передаются подпрограмме PDE_1D_MG как внешние подпрограммы. В качестве необязательного варианта их значения могут быть получены в результате дифференцирования назад (см. нижеприводимые примеры).

Подпрограмма PDE_1D_MG имеет вызов

CALL PDE_1D_MG(*t0, tout, ido, u,* [*initial_conditions,* &
pde_system_definition, boundary_conditions, iopt])

Параметры подпрограммы PDE_1D_MG разделяются на обязательные и необязательные. Последние в приведенном выше вызове заключены в квадратные скобки.

Обязательные параметры подпрограммы PDE_1D_MG:

Входной: tout.

Входные/выходные: t0, ido, u.

t0 - значение независимой переменной *t*, при котором начинается интегрирование u_t. Заменяется при завершении интегрирования на *tout*.

tout - значение независимой переменной *t*, при котором завершается интегрирование u_t. Заметьте, что допустимо задавать *t0* < *tout* и *t0* > *tout*. В зависимости от соотношения *t0* и *tout* меняется направление интегрирования.

ido - целочисленный флаг, управляющий программой. Принимает значения 1, 2, ..., 9. Причем значения *ido* = 5, 6, 7, 8, 9 используются приложением для информирования пользователя о возникающих проблемах и о ходе вычислений. Когда выполняется дифференцирование вперед, дифференциальные уравнения, граничные условия и начальные данные должны быть представлены в виде пользовательских подпрограмм. Отсутствие имен необязательных подпрограмм заставляет подпрограмму PDE_1D_MG прибегать к дифференцированию назад. Имена скаляров и массивов, хранящих результаты оценок, объявлены в модуле PDE_1D_MG_INT.MOD. Имена в зависимости от применяемой точности снабжены префиксами *s_pde_1d_mg_* или *d_pde_1d_mg_*. В качестве обобщенного обозначения далее используется префикс *?_pde_1d_mg_*.

Значения флага *ido* имеют следующий смысл:

- *ido* = 1 - задается перед первым вызовом подпрограммы. Обеспечивает выделение памяти в соответствии с размером задачи, а также иные инициализирующие действия;
- *ido* = 2 - это значение присваивается подпрограммой PDE_1D_MG, когда она благополучно завершила интегрирование, достигнув конечной точки *tout*;
- *ido* = 3 - задается пользователем после завершения интегрирования перед последним вызовом подпрограммы PDE_1D_MG;
- *ido* = 4 - возвращается решателем при возникновении фатальной или терминальной ошибки. Далее для корректного завершения работы подпрограммы PDE_1D_MG необходимо вызвать ее еще раз, задав прежде *ido* = 3;
- *ido* = 5 - возвращается решателем, когда необходимы начальные данные. После такого возврата и задания нужных данных решатель вызывается заново. Подготовка данных может включать обновление сетки, размещенной в сечении $u(npde + 1, 2{:}n - 1)$. Заметим, что пользователь получает равномерную сетку; при ее модификации значения в указанном сечении должны располагаться в возрастающем порядке. (Обновление сетки не обязательно.) В обязательном порядке перед новым вызовом PDE_1D_MG задаются значения всех компонентов системы в узлах сетки, размещенных в сечении $u(npde + 1, 1{:}n)$. Если же выполняется обновление сетки, то начальные значения оцениваются в новых узлах;
- *ido* = 6 - возвращается решателем, когда необходимы начальные данные для дифференциальных уравнений. После такого возврата и задания нужных данных решатель вызывается заново. Подготовка данных состоит в оценке функций системы (5.1). По умолчанию принимается, что $m = 0$. Это значение можно заменить на 1 или 2. Для этого используется вектор опций *iopt*. В скаляры и массивы заносятся следующие величины[1]:

$$x \equiv ?\_pde\_1d\_mg\_x;$$

$$t \equiv ?\_pde\_1d\_mg\_t;$$

$$u^j \equiv ?\_pde\_1d\_mg\_u(j);$$

$$\partial u^j / \partial x = u_x^j \equiv ?\_pde\_1d\_mg\_dudx(j);$$

[1] Знак присваивания, например $a := b$, используемый здесь и далее, означает, что оценивается выражение b и результат присваивается a.

$?\_pde\_1d\_mg\_c(j, k) := C_{jk}(x, t, u, u_x)$;

$?\_pde\_1d\_mg\_r(j) := r_j(x, t, u, u_x)$;

$?\_pde\_1d\_mg\_q(j) := q_j(x, t, u, u_x)$;

$j, k = 1, \ldots, npde$.

Если какая-нибудь функция не может быть оценена, задайте *pde_1d_mg_ires* = 3. В противном случае это значение оставьте неизменным;

- *ido*=7 - возвращается решателем, когда необходимы данные о граничных условиях (5.2). После такого возврата и задания нужных данных решатель вызывается заново. В скаляры и массивы, хранящие граничные условия, заносятся следующие величины:

  $x \equiv ?\_pde\_1d\_mg\_x$;

  $t \equiv ?\_pde\_1d\_mg\_t$;

  $u^j \equiv ?\_pde\_1d\_mg\_u(j)$;

  $\partial u^j / \partial x = u_x^j \equiv ?\_pde\_1d\_mg\_dudx(j)$;

  $?\_pde\_1d\_mg\_beta(j) := \beta_j(x, t, u, u_x)$;

  $?\_pde\_1d\_mg\_gamma(j) := \gamma_j(x, t, u, u_x)$;

  $j = 1, \ldots, npde$.

  Причем $x \in \{x_L, x_R\}$, а логический флаг *pde_1d_mg_left* равен .TRUE. при $x = x_L$ и равен .FALSE. при $x = x_R$. Если какая-нибудь функция не может быть оценена, задайте *pde_1d_mg_ires* = 3. В противном случае это значение оставьте неизменным;

- *ido* = 8 - возвращается решателем, когда в вызывающей программной единице необходимо подготовиться к решению ленточной линейной системы алгебраических уравнений. Это значение возникает, лишь когда в векторе *iopt* установлена опция *pde_1d_mg_rev_comm_factor_solve*, задающая дифференцирование назад. С этим флагом используются следующие определенные в файле PDE_1D_MG_INT.MOD переменные:

  pde_1d_mg_iband - ширина половины ленты симметрической ленточной матрицы (о способах задания ленточных матриц в программах см. в [5, разд. 4.3];

  pde_1d_mg_lda - ведущий размер ленточной матрицы, представляемой массивом *?_pde_1d_mg_a*, равный 3*pde_1d_mg_iband* + 1. Число уравнений в линейной системе *neq* = *n*(*npde* +1);

?*_pde_1d_mg_a* - массив формы (*pde_1d_mg_lda, neq*), содержащий ленточную симметрическую матрицу задачи;

pde_1d_mg_panic_flag - целочисленный флаг тревоги; устанавливается равным единице, если решаемая линейная система является вырожденной.

Код, реагирующий на флаг *ido* = 8, может быть, например, таким (см. также пример 5):

```
case(8)
  ! Разложение ленточной матрицы выполняется подпрограммой DL2CRB
  ! При желании пользователь может заменить эту подпрограмму на другую
  call dl2crb(neq, d_pde_1d_mg_a, pde_1d_mg_lda, pde_1d_mg_iband,          &
   pde_1d_mg_iband, d_pde_1d_mg_a, pde_1d_mg_lda, ipvt, rcond, work)
  if(rcond <= epsilon(one)) pde_1d_mg_panic_flag = 1
```

Значения используемых с флагом *ido* = 8 переменных, подаваемых на вход употребленной выше подпрограммы DL2CRB, определяются в подпрограмме PDE_1D_MG. Результат - полученное в DL2CRB разложение ленточной матрицы линейной системы передается также подпрограмме PDE_1D_MG (см. флаг *ido* = 9);

- *ido* = 9 - возвращается решателем, когда в вызывающей программной единице необходимо решить линейную систему с матрицей, разложение которой выполнено при обработке флага *ido* = 8. С флагом *ido* = 9 используются следующие определенные в файле PDE_1D_MG_INT.MOD переменные:

  ?*_pde_1d_mg_rhs* - вектор размера *neq*, содержащий правую часть линейной системы;

  pde_1d_mg_panic_flag - целочисленный флаг тревоги; устанавливается равным единице, если решаемая линейная система является вырожденной;

  ?*_pde_1d_mg_sol* - вектор размера *neq*, получающий решение линейной системы.

  Код, работающий с флагом *ido* = 9, может быть, например, таким (см. также пример 5):

```
case(9)
  ! Используем при решении факторизованную ранее ленточную матрицу
  call dlfsrb(neq, d_pde_1d_mg_a, pde_1d_mg_lda, pde_1d_mg_iband,          &
 pde_1d_mg_iband, ipvt, d_pde_1d_mg_rhs, 1, d_pde_1d_mg_sol)
```

u(1:*npde* + 1, 1:*n*) - перенимающий форму массив, содержащий входные данные о размерах задачи, граничных условиях и приблизительное решение системы. Размер задачи *npde* + 1 = SIZE(*u*, 1). Число точек сетки *n* = SIZE(*u*, 2).

Границы изменения переменной x задаются в массиве u следующим образом: $u(npde + 1, 1) = x_L$, $u(npde + 1, n) = x_R$. Не требуется определять элементы $u(npde + 1, 2{:}n - 1)$. При завершении вычислений элемент $u(i, j)$ массива $u(1{:}npde, 1{:}n)$ содержит приблизительное решение $u_i(x_j(tout), tout)$. Узел сетки $x_j(tout)$ хранится в элементе $u(npde + 1, j)$. Обычно в начале интегрирования узлы сетки размещаются равномерно. Расположение узлов сетки может быть задано подпрограммой *initial_conditions* или в процессе дифференцирования назад (см. флаг *ido*=5 и необязательный параметр *initial_conditions*).

Необязательные параметры подпрограммы PDE_1D_MG:

Все необязательные параметры подпрограммы PDE_1D_MG являются *входными.*

initial_conditions - имя внешней подпрограммы, создаваемой пользователем, применяемой при дифференцировании вперед. Если параметр *initial_conditions* не используется, то необходимые данные предоставляются подпрограмме PDE_1D_MG в результате дифференцирования назад. В подпрограмме *initial_conditions* задаются необходимые для задачи начальные значения, существующие, когда независимая переменная равна $t0$. Также эта подпрограмма может снабдить PDE_1D_MG неравномерной начальной (в момент $t0$) сеткой. Подпрограмма имеет следующий интерфейс:

```
subroutine initial_conditions(npde, n, u)
 integer npde, n
 real(kind(t0)) u(:, :)
end subroutine initial_conditions
```

Необязательно задаваемая сетка размещается в сечении $u(npde + 1, 2{:}n - 1)$. Элементы сечения располагаются в возрастающем порядке. На входе сетка является равномерной, но может быть модифицирована внутри подпрограммы *initial_conditions*.

В обязательном порядке подпрограмма *initial_conditions* обеспечивает начальные значения элементам сечения $u(:, 1{:}n)$ для всех узлов сетки, размещенных в сечении $u(npde + 1, 1{:}n)$. Если же выполнена необязательная модификация сетки, то начальные значения оцениваются в новых узлах.

Подпрограмма *initial_conditions* в простейшем варианте может быть, например, такой:

```
! Задает начальные значения
subroutine initial_conditions(npde, npts, u)
 implicit none
 integer npde, npts
 real(kind(1d0)) u(npde + 1, npts)
```

```
  u(1, :) = 1d0
  u(2, :) = 0d0
end subroutine initial_conditions
```

pde_system_definition - имя внешней подпрограммы, создаваемой пользователем, применяемой при дифференцировании вперед. Подпрограмма задает систему дифференциальных уравнений (5.1); имеет следующий интерфейс:

```
subroutine pde_system_definition(t, x, npde, u, dudx, c, q, r, ires)
  integer npde, ires
  real(kind(t0)) t, x, u(:), dudx(:)
  real(kind(t0)) c(:, :), q(:), r(:)
end subroutine pde_system_definition
```

По умолчанию принимается, что $m = 0$. При необходимости, используя необязательный параметр *iopt*, можно задать $m = 1$ или $m = 2$. Значения в массивы подпрограммы *pde_system_definition* заносятся следующим образом:

$$u^j \equiv u(j);$$

$$\partial u^j / \partial x = u_x^j \equiv dudx(j);$$

$$c(j, k) := C_{jk}(x, t, u, u_x),$$

$$r(j) := r_j(x, t, u, u_x);$$

$$q(j) := q_j(x, t, u, u_x);$$

$$j, k = 1, \ldots, npde.$$

Если какая-либо функция не может быть оценена, задайте *ires* = 3. В противном случае значение *ires* изменять нельзя.

boundary_conditions - имя внешней подпрограммы, создаваемой пользователем, применяемой при дифференцировании вперед. Подпрограмма задает граничные условия (4.14); имеет следующий интерфейс:

```
subroutine bc_01(t, beta, gamma, u, dudx, npde, left, ires)
  integer npde, ires
  logical left
  real(kind(1d0)) t, beta(npde), gamma(npde), u(npde), dudx(npde)
end subroutine bc_01
```

Значения в массивы подпрограммы *boundary_conditions* заносятся следующим образом:

$$u^j \equiv u(j);$$

$$\partial u^j / \partial x = u_x^j \equiv dudx(j);$$

beta(j) := $\beta_j(x, t, u, u_x)$;

gamma(j) := $\gamma_j(x, t, u, u_x)$;

$j = 1, \ldots, npde$.

Причем $x \in \{x_L, x_R\}$, а логический флаг *pde_1d_mg_left* равен .TRUE. при $x = x_L$ и равен =.FALSE. при $x = x_R$.

iopt - массив производного типа, содержащий приводимые ниже опции; используется для пересылки необязательных данных подпрограмме PDE_1D_MG. Для употребления *iopt* необходимо выполнить ссылку

```
use error_option_packet
```

Некоторые варианты работы с *iopt* имеются в приводимых ниже примерах 2-8.

В общем случае можно задать следующие опции:

Префикс опции = ?	*Имя опции*	*Значение*
s_, d_	pde_1d_mg_cart_coordinates	1
" "	pde_1d_mg_cyl_coordinates	2
" "	pde_1d_mg_sph_coordinates	3
" "	pde_1d_mg_time_smoothing	4
" "	pde_1d_mg_spatial_smoothing	5
" "	pde_1d_mg_monitor_regularizing	6
" "	pde_1d_mg_relative_tolerance	7
" "	pde_1d_mg_absolute_tolerance	8
" "	pde_1d_mg_max_bdf_order	9
" "	pde_1d_mg_rev_comm_factor_solve	10
" "	pde_1d_mg_no_nullify_stack	11

iopt(*io*) = *pde_1d_mg_cart_coordinates* - задает значение $m = 0$ в уравнении (5.1); это значение установлено по умолчанию.

iopt(*io*) = *pde_1d_mg_cyl_coordinates* - задает значение $m = 1$ в уравнении (5.1).

iopt(*io*) = *pde_1d_mg_sph_coordinates* - задает значение $m = 2$ в уравнении (5.1).

iopt(*io*) = ?_options(*pde_1d_mg_time_smoothing, tau*) - задает значение рассмотренного ниже при описании задачи временного сглаживающего параметра $\tau \geq 0$. По умолчанию $\tau = 0$.

iopt(*io*) = ?_options(*pde_1d_mg_spatial_smoothing, kap*) - задает значение рассмотренного ниже при описании задачи пространственного сглаживающего параметра $\kappa \geq 0$. По умолчанию $\kappa = 2$.

iopt(*io*) = ?_options(*pde_1d_mg_monitor_smoothing, alph*) - задает значение приведенного ниже параметра $\alpha \geq 0$. По умолчанию $\alpha = 0.01$.

iopt(*io*) = ?_options(*pde_1d_mg_relative_tolerance, rtol*) - задает допуск *rtol* оценки относительной ошибки, используемый подпрограммой DASPG. По умолчанию *rtol* = 1E-2 при работе с одинарной точностью и *rtol* = 1D-4 при работе с двойной.

iopt(*io*) = ?_options(*pde_1d_mg_absolute_tolerance, atol*) - задает допуск *atol* оценки абсолютной ошибки, используемый подпрограммой DASPG. По умолчанию *atol* = 1E-2 при работе с одинарной точностью и *atol* = 1D-4 при работе с двойной.

iopt(*io*) = *pde_1d_mg_max_bdf_order*

и

iopt(*io*+1) = *maxbdf* - задают максимальный порядок применяемых подпрограммой DASPG формул дифференцирования назад. По умолчанию *maxbdf* = 2. Новое значение - это число между 1 и 5. В некоторых задачах изменение порядка формул приведет к повышению качества решения. В примерах, однако, используется установленное по умолчанию значение, поскольку при работе с формулами, порядок которых выше двух, в подпрограмме DASPG могут возникнуть некоторые проблемы с выборам размера шага.

iopt(*io*) = *pde_1d_mg_rev_comm_factor_solve* - программная единица, вызывающая подпрограмму PDE_1D_MG, будет решать ленточную линейную систему, возникающую в жесткой системе алгебраических дифференциальных уравнений. Подпрограмма PDE_1D_MG возвращает значения *ido* = 8 и 9, только когда задана эта опция.

iopt(*io*) = *pde_1d_mg_no_nullify_stack* - обеспечивает полную трассировку возникающих в процессе работы подпрограммы PDE_1D_MG фатальных и терминальных ошибок. С целью повышения эффективности подпрограмма PDE_1D_MG выполняет уничтожение стека при возникновении в ней останавливающих ее функционирование ошибок. Это действие осуществляется в результате внутреннего вызова

CALL *e1psh*("NULLIFY_STACK")

и может быть полезным при использовании дифференцирования назад. При этом установленный процесс обработки ошибок выполняется как положено, однако точная информация о месте возникновения ошибки выдаваться не будет. После завершения работы решателя доступ к стеку восстанавливается выполняемым в PDE_1D_MG вызовом

CALL *e1pop*("NULLIFY_STACK")

Задание опции *pde_1d_mg_no_nullify_stack* обеспечит полную трассировку ошибки.

Описание:

Уравнение $u_t = f(u, x, t)$, $x_L < x < x_R$, $t > t_0$ аппроксимируется на зависящей от времени сетке

$$x_L <= x_0 < x_i(t) < \ldots < x_{i+1}(t) < x_n = x_R.$$

Для этого соотношение

$$\frac{du}{dt} = u_t + u_x \frac{dx}{dt}$$

преобразовывается в

$$\frac{du}{dt} - u_x \frac{dx}{dt} = u_t = f(u, x, t).$$

Применение центральных разделенных разностей для представления u_x приводит к системе обыкновенных дифференциальных уравнений

$$\frac{dU_i}{dt} - \frac{\left(U_{i+1} - U_{i-1}\right)}{\left(x_{i+1} - x_{i-1}\right)} \frac{dx_i}{dt} = F_i,\ t > t_0,\ i = 1, \ldots, n. \qquad (5.3)$$

Члены U_i и F_i представляют соответственно приблизительное решение уравнения в частных производных и значение $f(u, x, t)$ в точке $(x, t) = (x_i(t), t)$. Приведенная схема обеспечивает второй порядок точности относительно пространственной переменной x.

Система обыкновенных дифференциальных уравнений (5.3) является недоопределенной, поэтому прибавляются дополнительные уравнения. Они содержат параметры, которые могут настраиваться пользователем. Более того, в сложных задачах подбор значений этих параметров, как правило, неизбежен. Итак, добавляются следующие уравнения[1]:

$$\Delta x_i = x_{i+1} - x_i;$$

$$\eta_i = (\Delta x_i)^{-1};$$

$$\mu_i = \eta_i - \kappa(\kappa + 1)(\eta_{i+1} - 2\eta_i + \eta_{i-1}),\ 0 \le i \le n;$$

$$\eta_{-1} \equiv \eta_0,\ \eta_{n+1} \equiv \eta_n.$$

[1] Знак тождественного равенства, например $a \equiv b$, означает, что либо a и b есть один и тот же объект, либо a и b совпадают по значению.

Величины η_i - это так называемые точки концентрации сетки, а $\kappa \geq 0$ является пространственным сглаживающим параметром. Теперь точки сетки определяются неявно, так что

$$\frac{\mu_{i-1} + \tau \dfrac{d\mu_{i-1}}{dt}}{M_{i-1}} = \frac{\mu_i + \tau \dfrac{d\mu_1}{dt}}{M_i}, 1 \leq i \leq n, \tag{5.4}$$

где $\tau \geq 0$ - временной сглаживающий параметр. Выбор больших значений τ приведет к фиксированной сетке. Увеличение τ по отношению к заданному по умолчанию значению позволяет избежать ошибок в точках пересечения сетки. Делитель в равенстве (5.4)

$$M_i^2 = \alpha + (npde)^{-1} \sum_{j=1}^{npde} \frac{\left(U_{i+1}^j - U_i^j\right)^2}{\left(\Delta x_i\right)^2}.$$

Величина κ определяет уровень кластеризации, или пространственного сглаживания, точек сетки. Уменьшение κ по отношению к заданному по умолчанию значению снижает степень пространственного сглаживания. Параметры M_i аппроксимируют длину и помогают задать форму сетки, или распределение x_i. Параметр τ предотвращает немедленное изменение сетки к новому состоянию, определяемому M_i, и, следовательно, позволяет избежать колебаний сетки, приводящих к большим относительным ошибкам. Это важно при работе с крутыми градиентами.

Объединение дифференциальных уравнений и сглаживающих равенств, представленных в дискретной форме, дает неявную систему дифференциальных уравнений

$$A(Y)\frac{dY}{dt} = L(Y);$$

$$Y = \left(U_1^1, \ldots, U_1^{npde}, x_1, \ldots, U_j^1, \ldots, U_j^{npde}, x_j, \ldots\right)^T.$$

Подобная система алгебраических дифференциальных уравнений часто является жесткой. Она решается с применением подпрограммы DASPG и ее процедур, включая D2SPG (см. описание DASPG). Отметьте, что подпрограмма DASPG используется решателем PDE_1D_MG до тех пор, пока DASPG не завершится с флагом *ido* = 3. Если подпрограмма DASPG нужна во время оценки дифференциальных уравнений или граничных условий, необходимы второй процессор и межпроцессорные обмены. Единственными опциями DASPG, устанавливаемыми подпрограммой PDE_1D_MG, являются максимальный порядок формул дифференцирования назад и допуски *atol*

и *rtol* оценок абсолютной и относительной ошибок. Порядок установки этих опций приведен при описании подпрограммы DASPG и подпрограмм IUMAG и SUMAG, управляющих опциями.

Обращение к подпрограмме PDE_1D_MG становится возможным после включения в код ссылки

```
use pde_1d_mg_int
```

Имя решателя PDE_1D_MG является родовым, т. е. в зависимости от типа входных данных им будет использована одинарная или двойная точность (последняя рекомендуется разработчиками).

Подпрограмма PDE_1D_MG вызывается, как правило, в следующем цикле:

```
ido = 1
do
 select case(ido)
 case(1) {Инициализация процесса интегрирования}
 case(2) {Сохраняем решение и обновляем t0 и tout}
  if(условие завершения интегрирования) ido = 3
 case(3) {Нормальное завершение интегрирования}
  exit
 case(4) {Завершение интегрирования по причине ошибок}
  exit
 case(5) {Оцениваем начальные данные}
 case(6) {Оцениваем дифференциальные уравнения}
 case(7) {Оцениваем граничные условия}
 case(8) {Подготовка к решению ленточной системы}
 case(9) {Решение ленточной системы}
 end select
 call pde_1d_mg(t0, tout, ido, u, initial_conditions,                    &
      pde_system_definition, boundary_conditions, iopt)
end do
```

5.2. ПРИМЕРЫ УПОТРЕБЛЕНИЯ ПОДПРОГРАММЫ PDE_1D_MG

5.2.1. ПРИМЕР 1. ЭЛЕКТРОДИНАМИЧЕСКАЯ МОДЕЛЬ

Рассматривается электродинамическая модель, заимствованная из [14]. Описывается системой

$$u_t = \varepsilon p u_{xx} - g(u - v);$$

$$v_t = p v_{xx} + g(u - v),$$

где $g(z) = \exp(\eta z/3) - \exp(-2\eta z/3)$;

$0 \le x \le 1$, $0 \le t \le 4$;

$u_x = 0$, $v = 0$ при $x = 0$;

$u = 1$, $v_x = 0$ при $x = 1$;

$\varepsilon = 0.143$, $p = 0.1743$, $\eta = 17.19$.

В приводимом ниже коде

$C = I_2$;

$m = 0$, $R_1 = \varepsilon p u_x$, $R_2 = p v_x$;

$Q_1 = g(u - v)$, $Q_2 = -Q_1$;

$u = 1$ и $v = 0$ при $t = 0$.

Граничные условия задачи:

$\beta_1 = 1$, $\beta_2 = 0$, $\gamma_1 = 0$, $\gamma_2 = v$ при $x = x_L = 0$;

$\beta_1 = 0$, $\beta_2 = 1$, $\gamma_1 = u - 1$, $\gamma_2 = 0$ при $x = x_R = 0$.

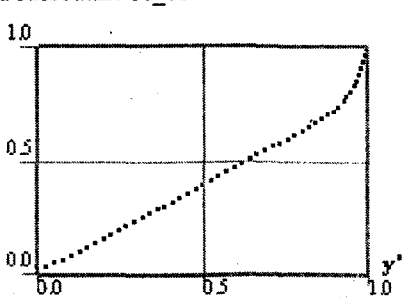
Обсуждение:

Рассматриваемая задача является нелинейной с резко изменяющимися условиями вблизи $t = 0$. Для ее решения, однако, достаточно использовать заданные по умолчанию настройки (параметры интегрирования). Применение в PDE_1D_MG дифференцирования вперед (выполняется по умолчанию) требует задания трех пользовательских подпрограмм, обеспечивающих задание начальных условий (подпрограмма *ic*_01), дифференциальных уравнений (подпрограмма *pde*_01) и граничных условий (подпрограмма *bc*_01).

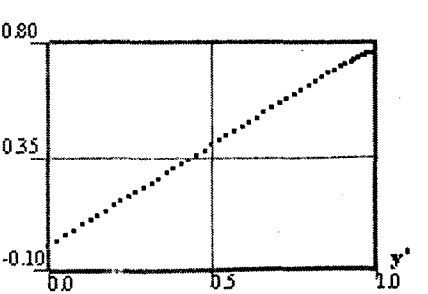
```
program pde_ex1                          ! Электродинамическая модель
 use pde_1d_mg_int
 implicit none
 integer, parameter :: npde = 2, n = 51, nframes = 5
 integer i, ido
 ! u - массив, содержащий решение
 real(kind(1d0)) u(npde + 1, n), t0, tout
 real(kind(1d0)) :: zero = 0d0, one = 1d0, delta_t = 10d0, tend = 4d0
 ! Массив, введенный для графического отображения
 ! результата - зависимостей u(x) и v(x) при t = tend
 real(4), allocatable :: uv(:, :)
 !dec$attributes array_visualizer :: uv
 external ic_01, pde_01, bc_01
 ido = 1                                 ! Начало интегрирования
 do
```

```
 select case(ido)
 case(1)                              ! Начальные данные
  t0 = zero; tout = 1d-3
  u(npde + 1, 1) = zero
  u(npde + 1, n) = one
  open(file = 'pde_ex01.out', unit = 7)
  write(7, "(3i5, 4f10.5)") npde, n, nframes, u(npde + 1, 1), u(npde + 1, n), t0, tend
 case(2)                              ! Переход к следующей выходной точке,
  write(7,"(f10.5)") tout             ! вывод решения, проверка на достижение
  do i = 1, npde + 1                  ! последней точки
   ! При завершении вычислений элемент u(i, j) массива u(1:npde, 1:n)
   ! содержит приблизительное решение u_i(x_j(tout), tout);
   ! узел сетки x_j(tout) хранится в элементе u(npde + 1, j),
   ! где i = 1, npde - номер зависимой переменной, а j = 1, n - номер узла сетки
    write(7,"(4e15.5)") u(i, :)
  end do
  t0 = tout; tout = tout * delta_t
  if(t0 >= tend) ido = 3
  tout = min(tout, tend)
 case(3)                              ! Интегрирование завершено
  close(unit = 7)
  allocate(uv(2, n))                  ! Графический вывод
  uv(1, :) = u(npde + 1, :)           ! Выводим график u(x, t = tend)
  uv(2, :) = u(1, :)
  ! Текст подпрограммы vGraph приведен в примере 1,
  ! сопровождающем описание подпрограммы DASPG (разд. 4.3)
  call vGraph(uv, n)                  ! Результат см. на рис. 5.1, а
  deallocate(uv)
  allocate(uv(2, n))
  uv(1, :) = u(npde + 1, :)           ! Выводим график v(x, t = tend)
  uv(2, :) = u(2, :)
  call vGraph(uv, n)                  ! Результат см. на рис. 5.1, б
  deallocate(uv)
  exit
 end select
 ! При интегрировании используется дифференцирование вперед
 call pde_1d_mg(t0, tout, ido, u, initial_conditions = ic_01,                    &
      pde_system_definition = pde_01, boundary_conditions = bc_01)
end do
end program pde_ex1

subroutine ic_01(npde, npts, u)       ! Задание начальных значений
 implicit none
```

```
 integer npde, npts
 real(kind(1d0)) u(npde + 1, npts)
 u(1, :) = 1d0; u(2, :) = 0d0
end subroutine ic_01

subroutine pde_01(t, x, npde, u, dudx, c, q, r, ires)
 implicit none                          ! Дифференциальные уравнения примера 1
 integer npde, ires
 real(kind(1d0)) t, x, u(npde), dudx(npde), c(npde, npde), q(npde), r(npde)
 real(kind(1d0)) :: eps = 0.143d0, p = 0.1743d0,                              &
      eta = 17.19d0, z, two = 2d0, three = 3d0
 c = 0d0; c(1, 1) = 1d0; c(2, 2) = 1d0
 r = p * dudx; r(1) = r(1) * eps
 z = eta * (u(1)-u(2)) / three
 q(1) = exp(z) - exp(-two * z)
 q(2) = -q(1)
end subroutine pde_01

subroutine bc_01(t, beta, gamma, u, dudx, npde, left, ires)
 implicit none                          ! Задает граничные условия примера 1
 integer npde, ires
 logical left
 real(kind(1d0)) t, beta(npde), gamma(npde), u(npde), dudx(npde)
 if(left) then
  beta(1) = 1d0;              beta(2) = 0d0
  gamma(1) = 0d0;             gamma(2) = u(2)
 else
  beta(1) = 0d0;              beta(2) = 1d0
  gamma(1) = u(1)-1d0;        gamma(2) = 0d0
 end if
end subroutine bc_01
```



а



б

Рис. 5.1. Графики функций: а - u(x, t = tend); б - v(x, t = tend)

Замечание. В этом и иных примерах результаты записываются в файлы PDE_ex0?.out. Полученные данные можно использовать для графического вывода результатов интегрирования, например посредством отображателя массивов.

5.2.2. ПРИМЕР 2. НЕВЯЗКИЙ ПОТОК НА ПЛАСТИНЕ

Невязкий поток на пластине описывается системой дифференциальных уравнений первого порядка [42]

$$u_t = -v_x;$$

$$uu_t = -vu_x + w_x;$$

$$w = u_x, \text{ т. е. } uu_t = -vu_x + u_{xx};$$

$$u(0, t) = v(0, t) = 0, u(\infty, t) \equiv u(x_R, t) = 1, t \geq 0;$$

$$u(x, 0) = 1, v(x, 0) = 0, x \geq 0.$$

Избавляясь от зависимой переменной w, получаем $npde = 2$ дифференциальных уравнений. Независимая переменная t не является временем, она суть вторая пространственная переменная. Интегрирование выполняется с $t = 0$ до $t = 5$. Конечное значение переменной x равно $x_{max} = x_R = 25$. Параметр $m = 0$ и

$$C = \{C_{jk}\} = \begin{pmatrix} 1 & 0 \\ u & 0 \end{pmatrix}, \quad R = \begin{pmatrix} -v \\ u_x \end{pmatrix}, \quad Q = \begin{pmatrix} 0 \\ vu_x \end{pmatrix}.$$

Граничные условия задаются равенствами

$$\beta = 0, \ \gamma = \begin{pmatrix} u - \exp(-20t) \\ v \end{pmatrix}$$

при $x = x_L$ и

$$\beta = 0, \ \gamma = \begin{pmatrix} u - 1 \\ v_x \end{pmatrix}$$

при $x = x_R$.

При интегрировании используется сетка, имеющая $n = 10 + 51 = 61$ точек, вывод результата выполняется с шагом $\Delta t = 01$.

Обсуждение:

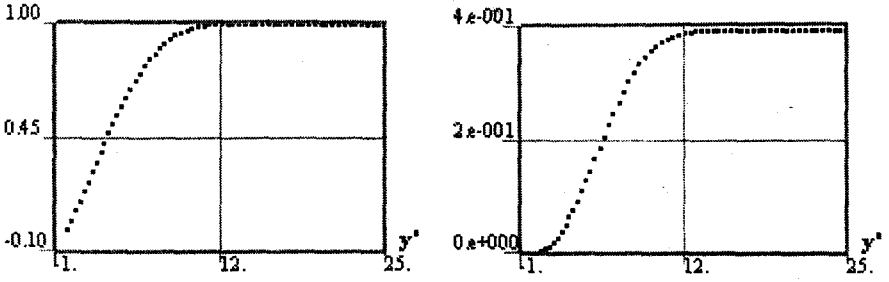
Рассматриваемая задача является нелинейной с резко изменяющимися условиями вблизи $t = 0$. Постановка задачи изменена таким образом, что граничные условия непрерывны около $t = 0$. Для этих целей подобрана функция $u - e^{-20t}$. Без таких изменений подпрограмма DASPG, вызываемая

подпрограммой PDE_1D_MG, не может решить задачу. (В изначальной постановке граничные условия имели разрыв при $t = 0$.) При интегрировании применяется дифференцирование назад, поэтому дополнительных пользовательских процедур, таких, как в примере 1, вводить не требуется: дифференциальное уравнение и граничные условия определяются в примере в главной программе. Предвидя быстрое изменение решения вблизи точки $x_L = 0$, в программе около x_L сосредоточено 10 начальных точек сетки. В качестве дополнительных необязательных настроек введены допуск на абсолютную ошибку и ненулевое значение временного сглаживающего параметра.

```
program pde_1d_mg_ex02                ! Невязкий поток на пластине
 use pde_1d_mg_int
 use error_option_packet
 implicit none
 integer, parameter :: npde = 2, n1 = 10, n2 = 51, n = n1 + n2
 integer i, ido, nframes
 real(kind(1d0)) u(npde + 1, n), t0, tout, dx1, dx2, diff
 real(kind(1d0)) :: zero = 0d0, one = 1d0, delta_t = 1d-1, tend = 5d0, xmax = 25d0
 real(kind(1d0)) :: u0 = 1d0, u1 = 0d0, tdelta = 1d-1, tol = 1d-2
 type(d_options) iopt(3)
 ! Массив, введенный для графического отображения
 ! результата - зависимостей u(x) и v(x) при t = tend
 real(4), allocatable :: uv(:, :)
 !dec$attributes array_visualizer :: uv
 ido = 1                              ! Начало интегрирования
 do
  select case(ido)
  case(1)                             ! Начальные данные; задание опций
   t0 = zero; tout = delta_t
   u(npde + 1, 1) = zero
   u(npde + 1, n) = xmax
   open(file = 'pde_ex02.out', unit = 7)
   nframes = nint((tend + delta_t) / delta_t)
   write(7, "(3i5, 4d14.5)") npde, n, nframes, u(npde + 1, 1), u(npde + 1, n), t0, tend
   dx1 = xmax / n2
   dx2 = dx1 / n1
   iopt(1) = d_options(pde_1d_mg_relative_tolerance, zero)
   iopt(2) = d_options(pde_1d_mg_absolute_tolerance, tol)
   iopt(3) = d_options(pde_1d_mg_time_smoothing, 1d-3)
  case(2)                             ! Переход к следующей выходной точке,
   t0 = tout                          ! вывод решения, проверка на достижение
   if(t0 <= tend) then                ! последней точки
    write(7, "(f10.5)") tout
```

```
  do i = 1, npde + 1
    write(7, "(4e15.5)") u(i, :)
  end do
  tout = min(tout + delta_t, tend)
  if(t0 == tend) ido = 3
 end if
case(3)                                  ! Интегрирование завершено
 close(unit = 7)
 ! Графический вывод
 allocate(uv(2, n))
 uv(1, :) = u(npde + 1, :)
 uv(2, :) = u(1, :)                      ! Выводим график u(x, t = tend)
! Текст подпрограммы vGraph приведен в примере 1,
! сопровождающем описание подпрограммы DASPG (разд. 4.3)
 call vGraph(uv, n)                      ! Результат см. на рис. 5.2, а
 deallocate(uv)
 allocate(uv(2, n))
 uv(1, :) = u(npde + 1, :)               ! Выводим график v(x, t = tend)
 uv(2, :) = u(2, :)
 call vGraph(uv, n)                      ! Результат см. на рис. 5.2, б
 deallocate(uv)
 exit
case(5)                                  ! Задаем начальные данные
 u(:npde, :) = zero; u(1, :) = one
 do i = 1, n1
  u(npde + 1, i) = (i - 1) * dx2
 end do
 do i = n1 + 1, n
  u(npde + 1, i) = (i - n1) * dx1
 end do
 write(7, "(f10.5)") t0
 do i = 1, npde + 1
  write(7, "(4e15.5)") u(i, :)
 end do
case(6)                                  ! Задаем дифференциальные уравнения
 d_pde_1d_mg_c = zero
 d_pde_1d_mg_c(1, 1) = one
 d_pde_1d_mg_c(2, 1) = d_pde_1d_mg_u(1)
 d_pde_1d_mg_r(1) = -d_pde_1d_mg_u(2)
 d_pde_1d_mg_r(2) = d_pde_1d_mg_dudx(1)
 d_pde_1d_mg_q(1) = zero
 d_pde_1d_mg_q(2) = d_pde_1d_mg_u(2) * d_pde_1d_mg_dudx(1)
case(7)                                  ! Задаем граничные условия
```

```
  d_pde_1d_mg_beta = zero
  if(pde_1d_mg_left) then
    diff = exp(-20d0 * d_pde_1d_mg_t)
    ! Обеспечиваем непрерывность граничных условий
    d_pde_1d_mg_gamma = (/ d_pde_1d_mg_u(1) - diff, d_pde_1d_mg_u(2) /)
  else
    d_pde_1d_mg_gamma = (/ d_pde_1d_mg_u(1) - one, d_pde_1d_mg_dudx(2) /)
  end if
 end select
 ! Применяем дифференцирование назад
 call pde_1d_mg(t0, tout, ido, u, iopt = iopt)
end do
end program pde_1d_mg_ex02
```



а *б*

Рис. 5.2. Графики функций: а - u(x, t = tend); б - v(x, t = tend)

5.2.3. ПРИМЕР 3. ДИНАМИКА ИЗМЕНЕНИЯ ЧИСЛЕННОСТИ НАСЕЛЕНИЯ

Динамика изменения численности населения моделируется системой [42]:

$$u_t = -u_x - I(t)u,\ x_L = 0 \le x \le a = x_R,\ t \ge 0;$$

$$I(t) = \int_0^a u(x,t)dx;$$

$$u(x,0) = \frac{\exp(-x)}{2 - \exp(-a)};$$

$$u(0,t) = g\left(\int_0^a b\left(x, I(t)\right)\ u(x,t)dx, t\right),$$

где

$$b(x,y)=\frac{xy\exp(-x)}{(y+1)^2}$$

и

$$g(z,t)=\frac{4z\left(2-2\exp(-a)+\exp(-t)\right)^2}{\left(1-\exp(-a)\right)\left(1-(1+2a)\exp(-2a)\right)\left(1-\exp(-a)+\exp(-t)\right)}.$$

Эта задача содержит определенную во всей области неизвестную функцию

$$u(x,t)=\frac{\exp(-x)}{1-\exp(-a)+\exp(-t)},$$

что делает проблему весьма примечательной. Решение задачи посредством подпрограммы PDE_1D_MG становится возможным после введения двух уравнений

$$v_1(t)=\int_0^a u(x,t)dx;$$

$$v_2(t)=\int_0^a x\exp(-x)u(x,t)dx,$$

что позволяет получить модифицированную систему

$$u_t=-u_x-v_1u,\ 0\le x\le a,\ t\ge 0;$$

$$u(0,\ t)=\frac{g(1,\ t)\ \ v_1v_2}{(v_1+1)^2}.$$

Таким образом, при решении дифференциального уравнения необходимо оценивать интегралы. Их оценка выполняется на той же сетке, что используется и для решения уравнения. Интегралы вычисляются по формуле трапеций, точность которой соизмерима с точностью, обеспечиваемой решателем PDE_1D_MG.

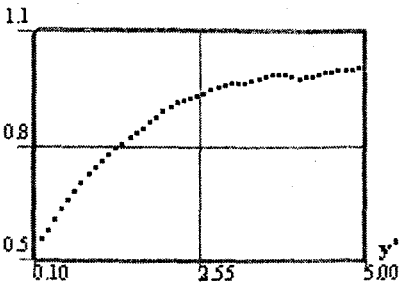
Обсуждение:

Настоящая задача является нелинейной интегрально-дифференциальной, содержащей нелокальные условия для дифференциального уравнения, а также граничные условия. Оценка этих условий может быть выполнена только при помощи дифференцирования назад. В качестве дополнительных необязательных настроек введены допуск на абсолютную ошибку и временной сглаживающий параметр $\tau = 1$. Это значение τ позволяет предотвратить пересечение линий сетки.

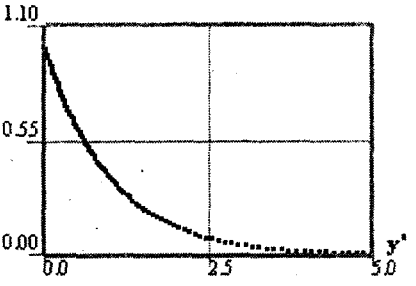
```
program pde_1d_mg_ex03                ! Модель роста численности населения
 use pde_1d_mg_int
 use error_option_packet
 implicit none
 integer, parameter :: npde = 1, n = 101
 integer ido, i, nframes, kt          ! kt - число шагов по переменной t
real(kind(1d0)) u(npde + 1, n), mid(n - 1), t0, tout, v_1, v_2
real(kind(1d0)) :: zero = 0d0, half = 5d-1, one = 1d0,                    &
      two = 2d0, four = 4d0, delta_t = 1d-1, tend = 5d0, a = 5d0
type(d_options) iopt(3)
! Массив, введенный для графического отображения
! результата - зависимости u(x) при t = tend
real(4), allocatable :: uv(:, :)
!dec$attributes array_visualizer :: uv
allocate(uv(2, int(tend / delta_t) + 1))
kt = 0; ido = 1                       ! Начало интегрирования
do
 select case(ido)
 case(1)                              ! Начальные данные; задание опций
  t0 = zero; tout = delta_t
  u(npde + 1, 1) = zero
  u(npde + 1, n) = a
  open(file = 'pde_ex03.out', unit = 7)
  nframes = nint((tend + delta_t) / delta_t)
  write(7, "(3i5, 4d14.5)") npde, n, nframes, u(npde + 1, 1), u(npde + 1, n), t0, tend
  iopt(1) = d_options(pde_1d_mg_relative_tolerance, zero)
  iopt(2) = d_options(pde_1d_mg_absolute_tolerance, 1d-2)
  iopt(3) = d_options(pde_1d_mg_time_smoothing, 1d0)
 case(2)                              ! Переход к следующей выходной точке,
  t0 = tout                           ! вывод решения, проверка на достижение
  if(t0 <= tend) then                 ! последней точки
   kt = kt + 1
   uv(1, kt) = t0
   uv(2, kt) = u(1, 1)
   write(7, "(f10.5)") tout
   do i = 1, npde + 1
    write(7, "(4e15.5)") u(i, :)
   end do
   tout = min(tout + delta_t, tend)
   if(t0 == tend) ido = 3
  end if
 case(3)                              ! Интегрирование завершено
  close(unit = 7)
```

```
 ! Графический вывод
 ! Прежде строим график u(x = 0, t)
 ! Текст подпрограммы vGraph приведен в примере 1,
 ! сопровождающем описание подпрограммы DASPG (разд. 4.3)
 call vGraph(uv, kt)                    ! Результат см. на рис. 5.3, а
 deallocate(uv)
 allocate(uv(2, n))
 uv(1, :) = u(npde + 1, :)              ! Выводим график u(x, t = tend)
 uv(2, :) = u(1, :)
 call vGraph(uv, n)                     ! Результат см. на рис. 5.3, б
 deallocate(uv)
 allocate(uv(2, n))
 exit
case(5)                                 ! Задаем начальные данные
 u(1, :) = exp(-u(2, :)) / (two - exp(-a))
 write(7, "(f10.5)") t0
 do i = 1, npde + 1
  write(7, "(4e15.5)") u(i, :)
 end do
case(6)                                 ! Задаем дифференциальные уравнения
 d_pde_1d_mg_c(1, 1) = one
 d_pde_1d_mg_r(1) = -d_pde_1d_mg_u(1)
 ! Оценка аппроксимации интеграла для момента времени t
 v_1 = half * sum((u(1, 1:n - 1) + u(1,2:n)) * (u(2, 2:n) - u(2,1:n - 1)))
 d_pde_1d_mg_q(1) = v_1 * d_pde_1d_mg_u(1)
case(7)                                 ! Задаем граничные условия
 if(pde_1d_mg_left) then
  ! Оценка аппроксимации интеграла для момента времени t;
  ! второй интеграл необходим на границе
  v_1 = half * sum((u(1, 1:n - 1) + u(1,2:n)) * (u(2, 2:n) - u(2,1:n - 1)))
  mid = half * (u(2, 2:n) + u(2,1:n - 1))
  v_2 = half * sum(mid * exp(-mid) * (u(1, 1:n - 1) + u(1, 2:n)) * (u(2, 2:n)-u(2,1:n - 1)))
  d_pde_1d_mg_beta = zero
  d_pde_1d_mg_gamma = g(one,d_pde_1d_mg_t) *                              &
     v_1 * v_2 / (v_1 + one)**2 - d_pde_1d_mg_u
 else
  d_pde_1d_mg_beta = zero
  d_pde_1d_mg_gamma = d_pde_1d_mg_dudx(1)
 end if
end select
! Употребляем дифференцирование назад
call pde_1d_mg(t0, tout, ido, u, iopt = iopt)
end do
```

```
contains
 function g(z, t)
  implicit none
  real(kind(1d0)) z, t, g
  g = four * z * (two - two * exp(-a) + exp(-t))**2
  g = g / ((one-exp(-a)) * (one - (one + two * a) * exp(-two * a)) * (1 - exp(-a) + exp(-t)))
 end function g
end program pde_1d_mg_ex03
```



а



б

Рис. 5.3. Графики функций: а - u(x = 0, t); б - u(x, t = tend)

5.2.4. ПРИМЕР 4. ДИФФУЗИЯ В РЕАКТОРЕ. МОДЕЛЬ В ЦИЛИНДРИЧЕСКИХ КООРДИНАТАХ

Задача взята из [14]. Диффузия моделируется уравнением

$$T_z = r^{-1}\frac{\partial\ (\beta r T_r)}{\partial\ r} + \gamma \exp\left(\frac{T}{1+\varepsilon T}\right);$$

$$T_r(0, z) = 0,\ T(1, z) = 0,\ z > 0;$$

$$T(r, 0) = 0,\ 0 \le r < 1;$$

$$\beta = 10^{-4},\ \gamma = 1,\ \varepsilon = 0.1.$$

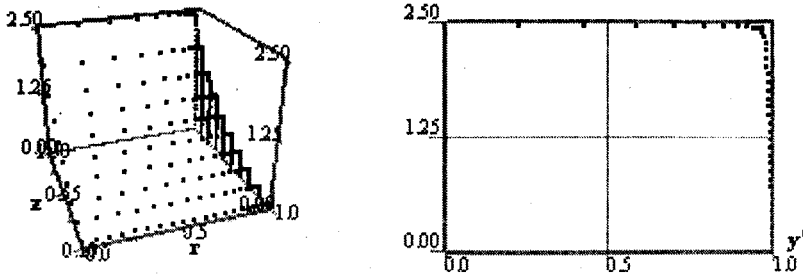
Осевое направление z рассматривается как временная координата, радиус r - как пространственная переменная.

Обсуждение:

Решается нелинейная задача, данная в цилиндрических координатах. Для работы с такими координатами задается опция $m = 1$. При вычислениях используется дифференцирование назад.

```
program pde_1d_mg_ex04
 use pde_1d_mg_int
 use error_option_packet
 implicit none
 integer, parameter :: npde = 1, n = 41
 integer ido, i, nframes, kz                ! kz - число шагов по переменной z
 real(kind(1d0)) T(npde + 1, n), z0, zout
 real(kind(1d0)) :: zero = 0d0, one = 1d0, delta_z = 1d-1,                    &
       zend = 1d0, zmax = 1d0, beta = 1d-4, gamma = 1d0, eps = 1d-1
 type(d_options) iopt(1)
 ! Массив, введенный для графического отображения
 ! результата - зависимостей T(r, z = zend) и T(r, z)
 real(4), allocatable :: Trz(:, :)
 !dec$attributes array_visualizer :: Trz
 allocate(Trz(3, n * int(zend / delta_z) + n))
 kz = 0; ido = 1                            ! Начало интегрирования
 do
  select case(ido)
  case(1)                                   ! Начальные данные; задание опции m = 1
   z0 = zero; zout = delta_z
   T(npde + 1, 1) = zero
   T(npde + 1, n) = zmax
   open(file = 'pde_ex04.out', unit = 7)
   nframes = nint((zend + delta_z) / delta_z)
   write(7, "(3i5, 4d14.5)")                                                  &
       npde, n, nframes, T(npde + 1, 1), T(npde + 1, n), z0, zend
   iopt(1) = pde_1d_mg_cyl_coordinates
  case(2)                                   ! Переход к следующей выходной точке,
   if(z0 <= zend) then                      ! вывод решения, проверка на достижение
    write(7, "(f10.5)") zout                ! последней точки
    do i = 1, npde + 1
     write(7, "(4e15.5)") T(i, :)
    end do
    kz = kz + n                             ! Готовим данные для графического вывода
    Trz(1, kz - n + 1:kz) = T(npde + 1, :)
    Trz(2, kz - n + 1:kz) = zout
    Trz(3, kz - n + 1:kz) = T(1, :)
    zout = min(zout + delta_z, zend)
    if(z0 == zend) ido = 3
   end if
  case(3)                                   ! Интегрирование завершено
   close(unit = 7)
   ! Графический вывод; выводим графики T(r, z) и T(r, z = zend)
```

```
 ! Текст подпрограммы vGraph2 приведен в разд. 4.6.1,
 ! содержащем описание подпрограммы FPS2H
 ! Результат см. на рис. 5.4, а
 call vGraph2(Trz, n * int(zend / delta_z) + n)
 deallocate(Trz)
 allocate(Trz(2, n))
 Trz(1, :) = T(npde + 1, :)
 Trz(2, :) = T(1, :)
 ! Текст подпрограммы vGraph приведен в примере 1,
 ! сопровождающем описание подпрограммы DASPG
 call vGraph(Trz, n)                    ! Результат см. на рис. 5.4, б
 deallocate(Trz)
 exit
case(5)                                 ! Задаем начальные данные
 T(1, :) = zero
 write(7, "(f10.5)") z0
 do i = 1, npde + 1
   write(7, "(4e15.5)") T(i, :)
 end do
case(6)                                 ! Задаем дифференциальные уравнения
 d_pde_1d_mg_c(1, 1) = one
 d_pde_1d_mg_r(1) = beta * d_pde_1d_mg_dudx(1)
 d_pde_1d_mg_q(1) = -gamma * exp(d_pde_1d_mg_u(1) /                    &
     (one + eps * d_pde_1d_mg_u(1)))
case(7)                                 ! Задаем граничные условия
 if(pde_1d_mg_left) then
   d_pde_1d_mg_beta = one
   d_pde_1d_mg_gamma = zero
 else
   d_pde_1d_mg_beta = zero
   d_pde_1d_mg_gamma = d_pde_1d_mg_u(1)
 end if
end select
! Употребляем дифференцирование назад;
! работаем с цилиндрическими координатами
call pde_1d_mg(z0, zout, ido, T, iopt = iopt)
end do
end program pde_1d_mg_ex04
```



а *б*

Рис. 5.4. Графики функций: а - T(r, z); б - T(r, z = zend)

5.2.5. ПРИМЕР 5. МОДЕЛЬ РАСПРОСТРАНЕНИЯ ПЛАМЕНИ

Пример взят из [54]. Задача нормализована. В системе, моделирующей распространение пламени, $u(x, t)$ - плотность массы, $v(x, t)$ - температура.

$$u_t = u_{xx} - uf(v);$$
$$v_t = v_{xx} + uf(v),$$

где

$$f(z) = \gamma\exp(-\beta / z),\ \beta = 4,\ \gamma = 3.52\times10^6;$$
$$0 \le x \le 1,\ 0 \le t \le 0.006;$$
$$u(x, 0) = 1,\ v(x, 0) = 0.2.$$

Граничные условия:

$$u_x = v_x = 0,\ x = 0;$$
$$u_x = 0,\ v_x = b(t),\ x = 1,$$

где

$$b(t) = \begin{cases} 1.2, & t \ge 2\times10^{-4}; \\ 0.2 + 5\times10^3 t, & 0 \le t < 2\times10^{-4}. \end{cases}$$

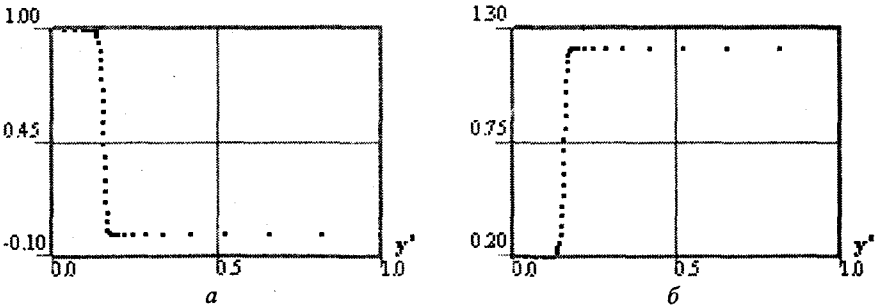
Обсуждение:

Решаемая задача является нелинейной. Пример показывает, как встроенный в подпрограмму PDE_1D_MG решатель линейных ленточных систем заменяется на аналогичный, но уже подобранный пользователем решатель. При вычислениях применяется дифференцирование назад. Линейная система объявляется вырожденной, если в процессе разложения ее матрицы выясняется, что величина, обратная числу обусловленности матрицы, меньше рабочей точности. Используемый в примере подход не следует распространять на все задачи, однако в случае его употребления следует фиксировать и соответствующим образом реагировать на факт вырожденности линейной системы.

```
program pde_1d_mg_ex05
 ! Модель распространения пламени
 use pde_1d_mg_int
 use error_option_packet
 use numerical_libraries, only : dl2crb, dlfsrb
 implicit none
 integer, parameter :: npde = 2, n = 40, neq = (npde + 1) * n
 integer i, ido, nframes, ipvt(neq)
 ! u - массив для решения исходного уравнения
 real(kind(1d0)) u(npde + 1, n), t0, tout
 ! Массив для решателя ленточных систем
 real(kind(1d0)) work(neq), rcond
 real(kind(1d0)) :: zero = 0d0, one = 1d0, delta_t = 1d-4, tend = 6d-3,          &
       xmax = 1d0, beta = 4d0, gamma = 3.52d6
 type(d_options) iopt(1)
 ! Массив, введенный для графического отображения
 ! результата - зависимостей u(x) и v(x) при t = tend
 real(4), allocatable :: uv(:, :)
 !dec$attributes array_visualizer :: uv
 ido = 1                                  ! Начало интегрирования
 do
  select case(ido)
  case(1)                                 ! Начальные данные
   t0 = zero
   tout = delta_t
   u(npde + 1, 1) = zero
   u(npde + 1, n) = xmax
   open(file = 'pde_ex05.out', unit = 7)
   nframes = nint((tend + delta_t) / delta_t)
   write(7, "(3i5, 4d14.5)")              &
        npde, n, nframes, u(npde + 1, 1), u(npde + 1, n), t0, tend
   iopt(1) = pde_1d_mg_rev_comm_factor_solve
  case(2)                                 ! Интегрирование
   t0 = tout
   if(t0 <= tend) then
    write(7,"(f10.5)") tout
    do i = 1, npde + 1
     write(7, "(4e15.5)") u(i, :)
    end do
    tout = min(tout + delta_t, tend)
    if(t0 == tend) ido = 3
   end if
  case(3)                                 ! Интегрирование завершено
```

```
 close(unit = 7)
 allocate(uv(2, n))                    ! Графический вывод
 uv(1, :) = u(npde + 1, :)             ! Выводим график u(x, t = tend)
 uv(2, :) = u(1, :)
 ! Текст подпрограммы vGraph приведен в примере 1,
 ! сопровождающем описание подпрограммы DASPG (разд. 4.3)
 call vGraph(uv, n)                    ! Результат см. на рис. 5.5, а
 deallocate(uv)
 allocate(uv(2, n))
 uv(1, :) = u(npde + 1, :)             ! Выводим график v(x, t = tend)
 uv(2, :) = u(2, :)
 call vGraph(uv, n)                    ! Результат см. на рис. 5.5, б
 deallocate(uv)
 exit
case(5)                                ! Задаем начальные данные
 u(1, :) = one
 u(2, :) = 2d-1
 write(7, "(f10.5)") t0
 do i = 1, npde + 1
  write(7, "(4e15.5)") u(i, :)
 end do
case(6)                                ! Задаем дифференциальные уравнения
 d_pde_1d_mg_c = zero
 d_pde_1d_mg_c(1, 1) = one
 d_pde_1d_mg_c(2, 2) = one
 d_pde_1d_mg_r = d_pde_1d_mg_dudx
 d_pde_1d_mg_q(1) = d_pde_1d_mg_u(1) * f(d_pde_1d_mg_u(2))
 d_pde_1d_mg_q(2) = -d_pde_1d_mg_q(1)
case(7)                                ! Задаем граничные условия
 if(pde_1d_mg_left) then
  d_pde_1d_mg_beta = zero
  d_pde_1d_mg_gamma = d_pde_1d_mg_dudx
 else
  d_pde_1d_mg_beta(1) = one
  d_pde_1d_mg_gamma(1) = zero
  d_pde_1d_mg_beta(2) = zero
  if(d_pde_1d_mg_t >= 2d-4) then
   d_pde_1d_mg_gamma(2) = 12d-1
  else
   d_pde_1d_mg_gamma(2) = 2d-1 + 5d3 * d_pde_1d_mg_t
  end if
  d_pde_1d_mg_gamma(2) = d_pde_1d_mg_gamma(2) - d_pde_1d_mg_u(2)
 end if
```

```
 case(8)
  ! Разложение ленточной матрицы. Используемый ниже решатель DL2CRB
  ! пользователь может заменить на другой
  call dl2crb(neq, d_pde_1d_mg_a, pde_1d_mg_lda, pde_1d_mg_iband,                &
      pde_1d_mg_iband, d_pde_1d_mg_a, pde_1d_mg_lda, ipvt, rcond, work)
  if(rcond <= epsilon(one)) pde_1d_mg_panic_flag = 1
 case(9)
 ! Используем при решении факторизованную ранее ленточную матрицу
 call dlfsrb(neq, d_pde_1d_mg_a, pde_1d_mg_lda, pde_1d_mg_iband,                 &
      pde_1d_mg_iband, ipvt, d_pde_1d_mg_rhs, 1, d_pde_1d_mg_sol)
 end select
 ! Употребляем дифференцирование назад
 call pde_1d_mg(t0, tout, ido, u, iopt = iopt)
end do

contains

function f(z)
 implicit none
 real(kind(1d0)) z, f
 f = gamma * exp(-beta / z)
end function f
end program pde_1d_mg_ex05
```



а *б*

Рис. 5.5. Графики функций: а - u(x, t = tend); б - v(x, t = tend)

5.2.6. ПРИМЕР 6. МОДЕЛЬ "ГОРЯЧЕЕ МЕСТО"

Пример описан в [54]. Приводимое ниже дифференциальное уравнение позволяет найти функцию температуры $u(x, t)$ в химической системе с реагирующими компонентами. Уравнение имеет вид:

$$u_t = u_{xx} + h(u),$$

где

$$h(z) = \frac{R}{a\delta}(1 + a - z)\exp\left(-\delta(1/z - 1)\right);$$

$a = 1, \delta = 20, R = 5;$

$0 \le x \le 1, 0 \le t \le 0.29;$

$u(x, 0) = 1;$

$u_x = 0, x = 0;$

$u = 1, x = 1.$

Обсуждение:

Задача является нелинейной. Быстро изменяющийся фронт, или "горячее место", обнаруживается после многих шагов интегрирования. Такой фронт вызывает быстрое изменение сетки. В программе задается опция, позволяющая использовать максимально возможный порядок формул дифференцирования назад, равный пяти. Напомним, что по умолчанию подпрограмма PDE_1D_MG работает с формулами дифференцирования назад 2-го порядка.

```
program pde_1d_mg_ex06
 use pde_1d_mg_int
 use error_option_packet
 implicit none
 integer, parameter :: npde = 1, n = 80
 integer i, ido, nframes
 real(kind(1d0)) u(npde + 1, n), t0, tout
 real(kind(1d0)) :: zero = 0d0, one = 1d0, delta_t = 1d-2,                    &
       tend = 29d-2, xmax = 1d0, a = 1d0, delta = 2d1, r = 5d0
 type(d_options) iopt(2)
 ! uv - массив, введенный для графического отображения
 ! результата - зависимости u(x) при t = tend
 real(4), allocatable :: uv(:, :)
 !dec$attributes array_visualizer :: uv
 ido = 1                                  ! Начало интегрирования
 do
  select case(ido)
  case(1)                                 ! Начальные данные; задание опций
   t0 = zero; tout = delta_t
   u(npde + 1, 1) = zero
   u(npde + 1, n) = xmax
   open(file = 'pde_ex06.out', unit = 7)
   nframes = (tend + delta_t) / delta_t
   write(7, "(3i5, 4d14.5)")              &
       npde, n, nframes, u(npde + 1, 1), u(npde + 1, n), t0, tend
```

```
 ! Теперь порядок формулы дифференцирования назад сможет увеличиваться
 ! до максимально допустимой величины, равной пяти
 iopt(1) = pde_1d_mg_max_bdf_order
 iopt(2) = 5
case(2)                              ! Переход к следующей выходной точке,
 t0 = tout                           ! вывод решения, проверка на достижение
 if(t0 <= tend) then                 ! последней точки
  write(7, "(f10.5)") tout
  do i = 1, npde + 1
   write(7, "(4e15.5)") u(i, :)
  end do
  tout = min(tout + delta_t, tend)
  if(t0 == tend) ido = 3
 end if
case(3)                              ! Интегрирование завершено
 close(unit = 7)
 allocate(uv(2, n))                  ! Графический вывод
 uv(1, :) = u(npde + 1, :)           ! Выводим график u(x, t = tend)
 uv(2, :) = u(1, :)
 ! Текст подпрограммы vGraph приведен в примере 1,
 ! сопровождающем описание подпрограммы DASPG (разд. 4.3)
 call vGraph(uv, n)                  ! Результат см. на рис. 5.6
 deallocate(uv)
 exit
case(5)                              ! Задаем начальные данные
 u(1, :) = one
 write(7, "(f10.5)") t0
 do i = 1, npde + 1
  write(7,"(4e15.5)") u(i, :)
 end do
case(6)                              ! Задаем дифференциальные уравнения
 d_pde_1d_mg_c = one
 d_pde_1d_mg_r = d_pde_1d_mg_dudx
 d_pde_1d_mg_q = -h(d_pde_1d_mg_u(1))
case(7)                              ! Задаем граничные условия
 if(pde_1d_mg_left) then
  d_pde_1d_mg_beta = zero
  d_pde_1d_mg_gamma = d_pde_1d_mg_dudx
 else
  d_pde_1d_mg_beta = zero
  d_pde_1d_mg_gamma = d_pde_1d_mg_u(1)-one
 end if
end select
```

```
 ! Употребляем дифференцирование назад
 call pde_1d_mg(t0, tout, ido, u, iopt = iopt)
 end do

contains

 function h(z)
  real(kind(1d0)) z, h
  h = (r/(a * delta)) * (one + a - z) * exp(-delta * (one / z-one))
 end function
end program pde_1d_mg_ex06
```

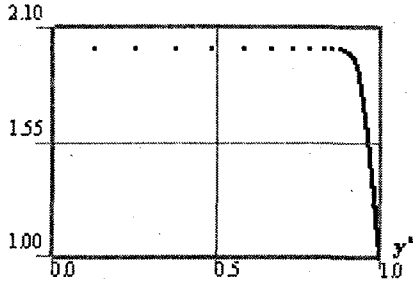


Рис. 5.6. График функции u(x, t = tend)

5.2.7. ПРИМЕР 7. БЕГУЩИЕ ВОЛНЫ

Пример описан в [54]. Решаемая система дифференциальных уравнений моделирует взаимодействие двух волн $u(x, t)$ и $v(x, t)$, движущихся в противоположных направлениях. Уравнения системы нормализованы. Волны встречаются, в результате чего их амплитуда уменьшается. Затем волны разделяются и перемещаются вперед, но с меньшей амплитудой. Система имеет вид:

$$u_t = -u_x - 100uv;$$

$$v_t = v_x - 100uv;$$

$$-0.5 \le x \le 0.5,\ 0 \le t \le 0.5;$$

$$u(x, 0) = 0.5(1 + \cos(10\pi x)),\ x \in [-0.3, -0.1] \text{ и } 0,$$

иначе

$$v(x, 0) = 0.5(1 + \cos(10\pi x)),\ x \in [0.3, 0.1] \text{ и } 0,$$

иначе

$$u = v = 0 \text{ на обоих концах, } t \ge 0.$$

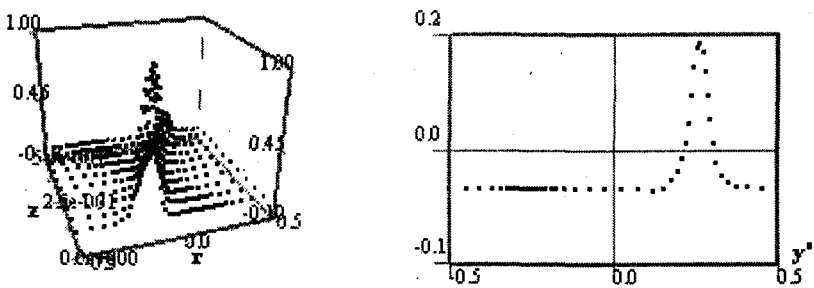
Обсуждение:

Система является нелинейной. Для ее решения употребляются формулы дифференцирования назад 3-го порядка и временной сглаживающий параметр $\tau = 0.001$.

```
program pde_1d_mg_ex07                ! Бегущие волны
 use pde_1d_mg_int
 use error_option_packet
 implicit none
 integer, parameter :: npde = 2, n = 50
 integer i, ido, nframes, kt          ! kt - число шагов по переменной t
real(kind(1d0)) u(npde + 1, n), temp(n), t0, tout
real(kind(1d0)) :: zero = 0d0, half = 5d-1, one = 1d0, delta_t = 5d-2,tend = 5d-1, pi
type(d_options) iopt(5)
! Массив, введенный для графического отображения
! результата - зависимостей u(x, t) и u(x, t = tend)
real(4), allocatable :: uxt(:, :)
!dec$attributes array_visualizer :: uxt
allocate(uxt(3, n * int(tend / delta_t) + n))
kt = 0; ido = 1
do
 select case(ido)
 case(1)                              ! Начальные данные; задание опций
  t0 = zero; tout = delta_t
  u(npde + 1, 1) = -half
  u(npde + 1, n) = half
  open(file = 'pde_ex07.out', unit = 7)
  nframes = (tend + delta_t) / delta_t
  write(7, "(3i5, 4d14.5)")                                          &
      npde, n, nframes, u(npde + 1, 1), u(npde + 1, n), t0, tend
  iopt(1) = d_options(pde_1d_mg_time_smoothing, 1d-3)
  iopt(2) = d_options(pde_1d_mg_relative_tolerance, zero)
  iopt(3) = d_options(pde_1d_mg_absolute_tolerance, 1d-3)
  iopt(4) = pde_1d_mg_max_bdf_order
  iopt(5) = 3
 case(2)                              ! Переход к следующей выходной точке,
  t0 = tout                           ! вывод решения, проверка на достижение
  if(t0 <= tend) then                 ! последней точки
   kt = kt + n                        ! Готовим данные для графического вывода
   uxt(1, kt- n + 1:kt) = u(npde + 1, :)
   uxt(2, kt - n + 1:kt) = t0
   uxt(3, kt - n + 1:kt) = u(1, :)
   write(7, "(f10.5)") tout
```

```
  do i = 1, npde + 1
    write(7, "(4e15.5)") u(i, :)
  end do
  tout = min(tout + delta_t, tend)
  if(t0 == tend) ido = 3
 end if
case(3)                                  ! Интегрирование завершено
 close(unit = 7)
 ! Графический вывод; выводим графики u(x, t) и u(x, t = tend)
 ! Текст подпрограммы vGraph2 приведен в разд. 4.6.1,
 ! содержащем описание подпрограммы FPS2H
 ! Результат см. на рис. 5.7, а
 call vGraph2(uxt, n*int(tend/delta_t)+n)
 deallocate(uxt)
 allocate(uxt(2, n))
 ! Выводим график u(x, t = tend)
 uxt(1, :) = u(npde + 1, :)
 uxt(2, :) = u(1, :)
 ! Текст подпрограммы vGraph приведен в примере 1,
 ! сопровождающем описание подпрограммы DASPG (разд. 4.3)
 call vGraph(uxt, n)                     ! Результат см. на рис. 5.7, б
 deallocate(uxt)
 exit
case(5)                                  ! Задаем начальные данные
 temp = u(3, :)
 u(1, :) = pulse(temp)
 u(2, :) = u(1, :)
 where(temp < -3d-1 .or. temp > -1d-1) u(1, :) = zero
 where(temp < 1d-1 .or. temp > 3d-1) u(2, :) = zero
 write(7, "(f10.5)") t0
 do i = 1, npde + 1
   write(7, "(4e15.5)") u(i, :)
 end do
case(6)                                  ! Задаем дифференциальные уравнения
 d_pde_1d_mg_c = zero
 d_pde_1d_mg_c(1, 1) = one
 d_pde_1d_mg_c(2, 2) = one
 d_pde_1d_mg_r = d_pde_1d_mg_u
 d_pde_1d_mg_r(1) = -d_pde_1d_mg_r(1)
 d_pde_1d_mg_q(1) = 100d0 * d_pde_1d_mg_u(1) * d_pde_1d_mg_u(2)
 d_pde_1d_mg_q(2) = d_pde_1d_mg_q(1)
case(7)                                  ! Задаем граничные условия
 d_pde_1d_mg_beta = zero
```

```
  d_pde_1d_mg_gamma = d_pde_1d_mg_u
 end select
 ! Употребляем дифференцирование назад
 call pde_1d_mg(t0, tout, ido, u, iopt = iopt)
end do

contains

function pulse(z)
 real(kind(1d0)) z(:), pulse(size(z))
 pi = acos(-one)
 pulse = half * (one + cos(10d0 * pi * z))
end function pulse
end program pde_1d_mg_ex07
```



а *б*

Рис. 5.7. Графики функций: а - u(x, t); б - u(x, t = tend)

5.2.8. ПРИМЕР 8. BLACK-SCHOLES

Величина европейского кол-опциона (т. е. опциона на покупку) $c(s, t)$ с ценой исполнения опциона e и датой завершения сделки T удовлетворяет условию

$$c(s, T) = \begin{cases} s, & s \geq e; \\ 0, & s < e. \end{cases}$$

До завершения опциона его величина $c(s, t)$ моделируется дифференциальным уравнением Black-Scholes

$$c_t + 0.5\sigma^2 s^2 c_{ss} + rsc_s - rc \equiv c_t + 0.5\sigma^2 (s^2 c_s)_s + (r - \sigma^2) sc_s - rc = 0.$$

Параметрами модели, приведенной в [56], являются процентный риск r, т. е. риск потерь или упущенной выгоды в связи с колебаниями процентных ставок и изменением стоимости кредитов, и непостоянство стоимости акций σ. В задаче следующие граничные условия:

$c(0, t) = 0$ и $c_s(s, t) \approx 1$ при $s \to 1$.

В примере принято, что

$$c = 100,\ r = 0.08,\ T - t = 0.25,\ \sigma^2 = 0.04,\ s_L = 0 \text{ и } s_R = 150.$$

Обсуждение:

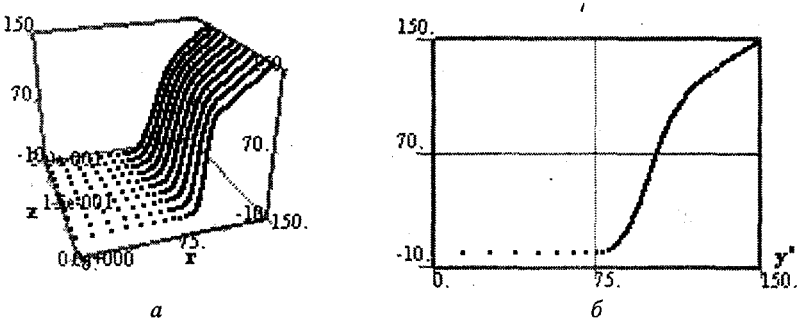
Решается линейная задача с разрывными начальными условиями. Для предотвращения пересечений линий сетки в задаче необходимо использовать временной сглаживающий параметр. Допуск оценки абсолютной ошибки задается равным 10^{-3} $US.

Существует явное решение этого уравнения, основанное на нормальном распределении вероятности. Нормальное распределение и само решение можно получить, применяя функцию ANORDF библиотеки IMSL. Численным интегрированием уравнения, дающим величину выплаты, легко манипулировать, изменяя формулу $c(s, T)$ и соответствующие граничные условия.

```
program pde_1d_mg_ex08                ! Black-scholes цена продаж
 use pde_1d_mg_int
 use error_option_packet
 implicit none
 integer, parameter :: npde = 1, n = 100
 integer i, ido, nframes, kt          ! kt - число шагов по переменной t
 real(kind(1d0)) u(npde + 1, n), t0, tout, sigsq, xval
 real(kind(1d0)) :: zero = 0d0, half = 5d-1, one = 1d0,                    &
       delta_t = 25d-3, tend = 25d-2, xmax = 150, sigma = 2d-1, r = 8d-2, e = 100d0
 type(d_options) iopt(5)
 ! Массив, введенный для графического отображения
 ! результата - зависимости u(x) при t = tend
 real(4), allocatable :: uxt(:, :)
 !dec$attributes array_visualizer :: uxt
 allocate(uxt(3, n * int(tend / delta_t) + n))
 kt = 0; ido = 1                      ! Начало интегрирования
 do
  select case(ido)
  case(1)                             ! Начальные данные; задание опций
   t0 = zero; tout = delta_t
   u(npde + 1, 1) = zero
   u(npde + 1, n) = xmax
   open(file = 'pde_ex08.out', unit = 7)
   nframes = nint((tend + delta_t) / delta_t)
   write(7, "(3i5, 4d14.5)") npde, n, nframes, u(npde + 1, 1), u(npde + 1, n), t0, tend
   sigsq = sigma**2
```

```
! Теперь порядок формулы дифференцирования назад сможет увеличиваться
! до максимально допустимой величины, равной пяти
iopt(1) = pde_1d_mg_max_bdf_order
iopt(2) = 5
iopt(3) = d_options(pde_1d_mg_time_smoothing, 5d-3)
iopt(4) = d_options(pde_1d_mg_relative_tolerance, zero)
iopt(5) = d_options(pde_1d_mg_absolute_tolerance, 1d-2)
case(2)                              ! Переход к следующей выходной точке,
t0 = tout                            ! вывод решения, проверка на достижение
if(t0 <= tend) then                  ! последней точки
  kt = kt + n                        ! Готовим данные для графического вывода
  uxt(1, kt - n + 1:kt) = u(npde + 1, :)
  uxt(2, kt - n + 1:kt) = tout
  uxt(3, kt - n + 1:kt) = u(1, :)
  write(7, "(f10.5)") tout
  do i = 1, npde + 1
    write(7,"(4e15.5)") u(i, :)
  end do
  tout = min(tout + delta_t, tend)
  if(t0 == tend) ido = 3
end if
case(3)                              ! Интегрирование завершено
close(unit = 7)
! Графический вывод; выводим графики u(x, t) и u(x, t = tend)
! Текст подпрограммы vGraph2 приведен в разд. 4.6.1,
! содержащем описание подпрограммы FPS2H
! Результат см. на рис. 5.8, а
call vGraph2(uxt, n*int(tend/delta_t)+n)
! Выводим график u(x, t = tend)
deallocate(uxt)
allocate(uxt(2, n))
uxt(1, :) = u(npde + 1, :)           ! Выводим график u(x, t = tend)
uxt(2, :) = u(1, :)
! Текст подпрограммы vGraph приведен в примере 1,
! сопровождающем описание подпрограммы DASPG (разд. 4.3)
call vGraph(uxt, n)                  ! Результат см. на рис. 5.8, б
deallocate(uxt)
exit
case(5)                              ! Задаем начальные данные
! Европейский кол-опцион - все или ничего
u(1, :) = max(u(npde + 1, :) - e, zero)
u(1, :) = u(npde + 1, :)
where(u(1, :) <= e) u(1, :) = zero
```

```
  write(7, "(f10.5)") t0
  do i = 1, npde + 1
    write(7, "(4e15.5)") u(i, :)
  end do
 case(6)                                  ! Задаем дифференциальные уравнения
  xval = d_pde_1d_mg_x
  d_pde_1d_mg_c = one
  d_pde_1d_mg_r = d_pde_1d_mg_dudx * xval**2 * sigsq * half
  d_pde_1d_mg_q = -(r-sigsq) * xval * d_pde_1d_mg_dudx + r * d_pde_1d_mg_u
 case(7)                                  ! Задаем граничные условия
  if(pde_1d_mg_left) then
    d_pde_1d_mg_beta = zero
    d_pde_1d_mg_gamma = d_pde_1d_mg_u
  else
    d_pde_1d_mg_beta = zero
    d_pde_1d_mg_gamma = d_pde_1d_mg_dudx(1) - one
  end if
 end select
 ! Употребляем дифференцирование назад
  call pde_1d_mg(t0, tout, ido, u, iopt = iopt)
end do
end program pde_1d_mg_ex08
```



а *б*

Рис. 5.8. Графики функций: а - u(x, t); б - u(x, t = tend)

ЛИТЕРАТУРА

1. Амосов А. А., Дубинский Ю. А., Копченова Н. В. Вычислительные методы для инженеров. - М.: Высш. шк., 1994. - 544 с.
2. Бартеньев О. В. Visual Fortran: Новые возможности. - М.: Диалог-МИФИ, 1999. - 288 с.
3. Он же. Графика OpenGL: программирование на Фортране. - М.: Диалог-МИФИ, 2000. - 368 с.
4. Он же. Современный Фортран. - М.: Диалог-МИФИ, 2000. - 448 с.
5. Он же. Фортран для профессионалов. Математическая библиотека IMSL: (Ч. 1). - М.: Диалог-МИФИ, 2000. - 448 с.
6. Боглаев Ю. П. Вычислительная математика и программирование. - М.: Высш. шк., 1990. - 544 с.
7. Де Бор К. Практическое руководство по сплайнам. - М.: Радио и связь, 1985. - 304 с.
8. Дэннис Дж., мл., Шнабель Р. Численные методы безусловной оптимизации и решения нелинейных уравнений. - М.: Мир, 1988. - 440 с.
9. Каханер Д., Моулер К., Нэш С. Численные методы и математическое обеспечение. - М.: Мир, 1988. - 575 с.
10. Самарский А. А. Введение в численные методы. - М.: Наука, 1997. - 239 с.
11. Эльсгольц Л. Э. Дифференциальные уравнения и вариационное исчисление. - М.: Наука, 1969. - 424 с.
12. Akima H. A new method of interpolation and smooth curve fitting based on local procedures// Journal of the ACM. 1970. N17. P. 589-602.
13. Akima H. A method of bivariate interpolation and smooth surface fitting for irregularly distributed data points, ACM Transactions on Mathematical Software, 4, 1978, 148-159.
14. Blom J. G., Zegeling P. A. Algorithm 731: A Moving-Grid Interface for Systems of One-Dimensional Time-Dependent Partial Differential Equations. ACM-Trans. Math. Soft., 20, 2, 194-214, 1994.
15. Boisvert R. A fourth order accurate fast direct method for the Helmholtz equation, Elliptic Problem Solvers II, Academic Press, Orlando, Florida, 35-44, 1984.
16. Brankin R. W., Gladwell I., Shampine L. F. RKSUITE: a Suite of Runge-Kutta Codes for the Initial Value Problem for ODEs, Softreport 91-1, Mathematics Department, Southern Methodist University, Dallas, Texas, 1991.
17. Brenan K. E., Campbell S. L., Petzold L. R. Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations. Elseview Science Publ. Co., 1989.
18. Cheney E.W. Introduction to Approximation Theory. New York: McGraw-Hill, 1966.
19. Cody W. J., Fraser W., Hart J. F. Rational Chebyshev approximation using linear equations, Numerische Mathematik, 12, 1968, 242-251.

20. Courant R., Hilbert D. Methods of Mathematical Physics. V. 2. New York: John Wiley & Sons, 1962.
21. Craven P., Wahba G. Smoothing noisy data with spline functions, Numerische Mathematik, 31, 1979, 377-403.
22. Davis P. F., Rabinowitz P. Methods of Numerical Integration. Orlando, Florida: Academic Press, 1984.
23. Draper N. R., Smith H. Applied Regression Analysis. 2nd ed. New York: John Wiley & Sons, 1981.
24. Enright W. H., Pryce J. D. Two FORTRAN packages for assessing initial value methods, ACM Transactions on Mathematical Software, 13, 1-22, 1987.
25. Forsythe G. E. Generation and use of orthogonal polynomials for fitting data with a digital computer, SIAM Journal on Applied Mathematics, 5, 1957, 74-88.
26. Franke R. Scattered Data Interpolation: Tests of Some Methods, Mathematics of Computation, 37, 157, 1982, 181-200.
27. Gautschi W. Construction of Gauss-Christoffel quadrature formulas. Mathematics of Computation, 22, 251-270, 1968.
28. Gautschi W., Milovanofic G. V. Gaussian quadrature involving Einstein and Fermi functions with an application to summation of series. Mathematics of Computation, 44, 177-190, 1985.
29. Gear C. W. Numerical Initial Value Problems in Ordinary Differential Equations. New Jersey: Prentice-Hall, Englewood Cliffs, 1971.
30. Gear C. W., Petzold Linda R. ODE methods for the solutions of differential/algebraic equations, SIAM Journal Numerical Analysis, 21, № 4, 716, 1984.
31. Golub G. H., Welsch J. H. Calculation of Gaussian quadrature rules. Mathematics of Computation, 23, 221-230, 1969.
32. Grosse E. Tensor spline approximation, Linear Algebra and its Applications, 34, 1980, 29-41.
33. Guerra V., Tapia R. A. A local procedure for error detection and data smoothing, MRC Technical Summary Report 1452, Mathematics Research Center, University of Wisconsin, Madison, 1974.
34. Hanson R. J. Constrained B-Spline Surface Fitting to Discrete Data, Technical Report 9503, Visual Numerics, Inc., Houston, Texas, 1995.
35. Hull T. E., Enright W. H., Jackson K. R. User's guide for DVERK - A subroutine for solving non-stiff ODEs, Department of Computer Science Technical Report 100, University of Toronto, 1976.
36. Irvine L. D., Marin S. P., Smith P. W. Constrained interpolation and smoothing, Constructive Approximation, 2, 1986, 129-151.
37. Kennedy W. J., Gentle J. E. Statistical Computing. New York: Marcel Dekker, 1980.
38. Lawson C. L., Hanson R. J. Solving Least Squares Problems, Classics in Applied Mathematics, 15, SIAM Publications, Philadelphia, PA, 1995.

39. Madsen N. K., Sincovec R. F. Algorithm 540: PDECOL, General collocation software for partial differential equations, ACM Transactions on Mathematical Software, 5, '3, 326-351, 1979.
40. Micchelli C. A., Smith P. W., Swetits J., Ward J. D. Constrained L S approximation, Constructive Approximation, 1, 1985, 93-102.
41. Neter J., Wasserman W. Applied Linear Statistical Models, Richard D. Irwin, Homewood, Ill, 1974.
42. Pennington S. V., Berzins M. New NAG Library Software for First Order Partial Differential Equations. ACM-Trans. Math. Soft., 20,1, 63-99, 1994.
43. Pereyra V. PASVA3: An adaptive finite-difference FORTRAN program for first order nonlinear boundary value problems, in Lecture Notes in Computer Science, 76, Springer-Verlag, Berlin, 67-88, 1978.
44. Petzold L. R. A description of DASSL: A differential/ algebraic system solver, Proceedings of the IMACS World Congress, Montreal, Canada, 1982.
45. Piessens R., deDoncker-Kapenga E., Uberhuber C. W., Kahaner D. K. QUADPACK. New York: Springer-Verlag, 1983.
46. Pruess S., Fulton C. T. Mathematical Software for Sturm-Liouville Problems. ACM Transactions on Mathematical Software, 17, 3, 360-376, 1993.
47. Reinsch C. H. Smoothing by spline functions, Numerische Mathematik, 10, 1967, 177-183.
48. Scott M. R., Shampine L. F., Wing G. M. Invariant Embedding and the Calculation of Eigenvalues for Sturm-Liouville Systems. Computing, 4, 10-23, 1969.
49. Sewell G. IMSL software for differential equations in one space variable, IMSL Technical Report 8202, IMSL, Houston, 1982.
50. Shampine L. F. Discrete least-squares polynomial fits, Communications of the ACM, 18, 1975, 179-180.
51. Shampine L. F., Gear C.W. A user's view of solving stiff ordinary differential equations, SIAM Review, 21, 1-17, 1979.
52. Sincovec R. F., Madsen N. K. Software for nonlinear partial differential equations, ACM Transactions on Mathematical Software, 1, N 3, 232-260, 1975.
53. Titchmarsh E. Eigenfunction Expansions Associated with Second Order Differential Equations. P. I, 2nd ed. London: Oxford University Press, 1962.
54. Verwer J. G., Blom J. G., Furzeland R. M., Zegeling P. A. A Moving-Grid Method for One-Dimensional PDEs Based on the Method of Lines. Adaptive Methods for Partial Differential Equations. Flaherty J. E., et al., Eds., SIAM Publications, Philadelphia, PA, 1989.
55. Washizu K. Variational Methods in Elasticity and Plasticity. New York: Pergamon Press, 1968.
56. Wilmott P., Howison S., Dewynne J. The Mathematics of Financial Derivatives. New York: Cambridge University Press, 1995.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

СОДЕРЖАНИЕ