Д. Д. Харлампиди
С. О. Адамсон

ВЫЧИСЛИТЕЛЬНЫЕ МЕТОДЫ В ХИМИИ

Москва
2020

Министерство просвещения Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский педагогический государственный университет»

Д. Д. Харлампиди, С. О. Адамсон

ВЫЧИСЛИТЕЛЬНЫЕ МЕТОДЫ В ХИМИИ

Учебное пособие

МПГУ
Москва • 2020

УДК 378(075.8):[54+004.9]
ББК 24с51я73
Х211

Рецензенты:

Н. Н. Камкин, кандидат химических наук,
доцент кафедры общей химии МПГУ

А. А. Беляев, кандидат физико-математических наук,
ведущий научный сотрудник ФИЦ ХФ РАН им. Н. Н. Семенова

Харлампиди, Дарья Дмитриевна.

Х211 Вычислительные методы в химии : учебное пособие / Д. Д. Харлампиди, С. О. Адамсон. – Москва : МПГУ, 2020. – 80 с.

ISBN 978-5-4263-0908-1

Пособие содержит рекомендации для самостоятельного выполнения практических заданий по дисциплине «Вычислительные методы в химии» с использованием языка программирования FORTRAN 77. Для каждого задания указаны необходимые для работы разделы учебников и содержание лекций. Для удобства выполнения заданий в первой главе книги приведены методические указания по программированию на языке FORTRAN 77. Пособие ориентировано на студентов и преподавателей педагогических университетов, работающих по направлениям бакалавриата и магистратуры 04.03.01 «Химия», 04.04.01 «Химия окружающей среды». Может быть также полезно и для педагогических направлений с профилем «Химия».

УДК 378(075.8):[54+004.9]
ББК 24с51я73

ISBN 978-5-4263-0908-1

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Развитие полупроводниковых технологий во второй половине XX века привело за относительно краткий промежуток времени к массовому внедрению электронных вычислительных машин (компьютеров) во многие области науки и техники. В химии полупроводниковая революция позволила не только упростить проведение эксперимента и ускорить обработку его результатов, но и заменить сложные и дорогостоящие эксперименты компьютерным моделированием. Из наиболее важных направлений компьютерного моделирования следует указать на компьютерный дизайн новых лекарственных средств, предсказание структуры и свойств как отдельных молекул, так и их агрегатов, конструирование материалов с требуемыми свойствами и т. д. Для понимания основ компьютерного эксперимента необходимо, с одной стороны, освоить методы построения компьютерных моделей, включая и особенности их применения, а с другой – способы их практической реализации на современных вычислительных машинах.

В настоящем пособии изложены основы алгоритмического языка FORTRAN 77 (глава 1), вместе с задачами, решение которых способствует освоению материала и иллюстрирует применение вычислительных методов к решению ряда прикладных задач физической химии (глава 2). Задачи представлены в формате практических работ, которые могут быть выполнены студентами самостоятельно или под руководством преподавателя, как в аудиторное, так и во внеаудиторное время. Выполнение практических заданий предлагается студентам проводить с помощью языка программирования FORTRAN, для которого на сегодняшний день имеется самая полная и проверенная годами база открытых алгоритмов по множеству химических и физических задач. Рекомендации по построению графиков функций даны на примере использования одного из простейших открытых графических редакторов XMGrace.

Пособие направлено на достижение целей освоения дисциплин: «Вычислительные методы в химии» (направление бакалавриата 04.03.01 «Химия») и «Вычислительные методы в химии и химическом мониторинге» (направление магистратуры 04.04.01 «Химия окружающей среды») и соответствует рабочим программам

этих дисциплин. Основными целями для бакалавриата являются: формирование у обучающихся способности анализировать и интерпретировать результаты химических экспериментов, наблюдений и измерений (ОПК-1), а также способности применять расчетно-теоретические методы для изучения свойств веществ и процессов с их участием с использованием современной вычислительной техники (ОПК-3); способности использовать существующие программные продукты и информационные базы данных для решения задач профессиональной деятельности с учетом основных требований информационной безопасности (ОПК-5). Основные цели для магистратуры: формирование у обучающихся способности выполнять комплексные экспериментальные и расчетно-теоретические исследования в избранной области химии или смежных наук с использованием современных приборов, программного обеспечения и баз данных профессионального назначения (ОПК-1); способности анализировать, интерпретировать и обобщать результаты экспериментальных и расчетно-теоретических работ в избранной области химии или смежных наук (ОПК-2); а также способности использовать вычислительные методы и адаптировать существующие программные продукты для решения задач профессиональной деятельности (ОПК-3).

ГЛАВА 1
КРАТКОЕ ОПИСАНИЕ ЯЗЫКА FORTRAN

1. Историческая справка

Язык программирования FORTRAN был разработан в 1954–1956 гг. группой сотрудников корпорации IBM под руководством Д. Бэкуса (John W. Backus, 3.12.1924–17.03.2007). В последующие годы появилось несколько программных диалектов языка, что затруднило межплатформный перенос программ. Для устранения этой проблемы в 1966 г. был выработан стандарт FORTRAN 66 (FORTRAN IV). На следующих этапах развития языка появились стандарты FORTRAN 77 (1978 г.), FORTRAN 90 (1990 г.) FORTRAN 95 (1997 г.), FORTRAN 2003 (2004 г.), FORTRAN 2008 (2010 г.) и FORTRAN 2015 (2018 г.). В России (СССР) FORTRAN начал применяться с конца 1960-х гг. Несколько позднее были разработаны отечественные стандарты его использования: ГОСТ 23056–78 (Фортран) и ГОСТ 23057–78 (Базисный Фортран).

Ниже в краткой форме излагаются базовые элементы языка FORTRAN 77, который достаточно прост и, несмотря на почтенный возраст, используется вплоть до сегодняшнего дня. В случае необходимости более поздние версии языка FORTRAN можно изучить самостоятельно.

2. Символы

Первичными элементами языка FORTRAN является набор *символов*, подразделяемых на *основные* и *дополнительные*. Набор основных символов строго определен и используется для образования всех конструкций языка. Набор дополнительных символов зависит от *платформы*, на которой реализуется язык, т. е. от типа *компьютера* (в отечественной терминологии – *электронной вычислительной машины*, сокращенно – ЭВМ), и установленной на нем операционной системы (ОС). Дополнительные символы используются при определении символьных констант.

Основные символы включают в себя *буквы*, *цифры* и *специальные символы*. В качестве букв взяты заглавные буквы латинского алфавита (A, B, C, D, E, F, H, G, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z); цифр – символы 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Специальные символы языка приведены в табл. 1.

Таблица 1

Специальные символы языка FORTRAN

Символ	Название символа	Символ	Название символа
␣	пробел	(	скобка левая (закрывающая)
=	равно	)	скобка правая (закрывающая)
+	плюс	,	запятая
-	минус	.	точка
*	звездочка	&	амперсенд (коммерческое «и»)
/	косая черта	'	апостроф

3. Операторы

Основными конструкциями языка являются *операторы*, подразделяемые на два класса: *выполняемые* и *невыполняемые*. Выполняемые операторы используются для указания действий и порядка их выполнения, невыполняемые – указывают на обозначения используемых величин, определения их типа и структуры. Любой из выполняемых операторов программы может быть *помечен* уникальным набором цифр. Этот цифровой набор принято называть *меткой* оператора. Некоторые из невыполняемых операторов также могут иметь метки.

4. Символические имена

Для обозначения обрабатываемых данных (независимых переменных, функций, элементов матриц, символьных последовательностей, названий программ и подпрограмм и т. д.) в языке

используются *символические имена* (иначе – *идентификаторы*). Символические имена являются не пустыми последовательностями букв и цифр, начинающимися с буквы и состоящими не более чем из шести символов.

5. Структура программы

Программа на языке FORTRAN представляет собой последовательность операторов. В общем случае программа включает *основную программу* и *подпрограммы*, вызываемые основной программой или другими подпрограммами. Составляющие (компоненты) программы принято называть *программными модулями* (*сегментами программы*). Выполнение программы всегда начинается с основной программы.

Преобразование программы на языке FORTRAN в исполняемую машиной программу проводится в несколько этапов. На первом этапе текст программы (*исходный код*) должен быть введен в память ЭВМ и сохранен на одном из ее запоминающих устройств. Далее при помощи специальной программы (*компилятора*) исходный код переводится в набор *объектных модулей*, которые собираются в исполняемую машиной программу (*исполняемый код*).

На ранних этапах развития вычислительных машин программы набирались на перфорированных картах (*перфокартах*), с которых загружались в память машины при помощи специализированного устройства ввода. Конструкция устройства позволяла считывать с одной перфокарты не более одной строки длиной до 80 символов, что и определило способ записи операторов в языке FORTRAN. Строка из 80 символов разбивается на 3 поля: 1–5-я позиции должны содержать метку или пробелы в случае ее отсутствия, в 7–72-й позициях записывался оператор. В том случае, если в 6-й позиции вводится символ, отличный от пробела, то текущая строка является продолжением оператора, введенного в предыдущих строках (*строкой продолжения оператора*), в противном случае – новым оператором (*начальной строкой оператора*). Оператор может состоять не более чем из 20 строк, причем начальная строка всегда следует первой, а за ней следуют строки

продолжения. При необходимости программа снабжается строками комментариев, начинающихся с символа «*» (звездочка) или буквы «C». Строки комментариев могут находиться в любом месте программы.

Основная программа на языке FORTRAN начинается с определения ее названия (*заголовка программы*). Заголовок имеет вид

PROGRAM ***name***

где ***name*** – собственно название программы, начинающееся с буквы и состоящее не более чем из 8 основных символов. В строках программы, следующих за заголовком, должны находиться невыполняемые операторы, описывающие используемые программой переменные. После них должны следовать выполняемые операторы, указывающие на манипуляции, проводимые с описанными выше переменными. После выполняемых операторов должен следовать оператор *останова*, имеющий вид

STOP ***m***

где ***m*** – целое положительное число или пробел. При необходимости оператору *останова* может быть присвоена метка. Если ***m*** не является пробелом, то при выполнении данного оператора на консоль выдается сообщение 'STOP ***m***'. Последним оператором программы должен быть оператор конца

END

Подпрограммы подразделяются на *подпрограммы-процедуры* и *подпрограммы-функции* (иногда говорят о внешних процедурах и внешних функциях). Подпрограмма-процедура начинается с заголовка

SUBROUTINE ***name*** (***parameters***)

и завершается, как и основная программа, оператором END. В заголовке ***name*** – название подпрограммы, а ***parameters*** – список *формальных параметров* подпрограммы, являющихся символическими именами. Обращение к подпрограмме осуществляется с помощью оператора CALL:

CALL ***name*** (***parameters***)

где ***name*** – список *фактических параметров*.

Подпрограмма-функция начинается с заголовка

type FUNCTION ***name*** (***parameters***)

и завершается оператором конца END. В заголовке подпрограммы-функции ***name*** – название подпрограммы-функции, ***parameters*** – список формальных параметров, а ***type*** – тип функ-

ции (может быть опущен). Обращение к подпрограмме-функции производится при помощи конструкции

name (***parameters***)

При обращении к обоим типам подпрограмм формальные параметры из списка в круглых скобках заменяются фактическими параметрами из основной программы или других подпрограмм (*вызывающих сегментов*). При этом количество, порядок расположения и типы параметров в списках должны совпадать. Порядок размещения выполняемых и невыполняемых операторов в подпрограммах такой же, как и в основной программе; т. е. невыполняемые операторы следуют сразу за заголовком и только потом идут выполняемые операторы. Для передачи управления из подпрограммы в вызывающий сегмент необходимо использовать *оператор возврата*, имеющий вид

RETURN

В случае его отсутствия подпрограмма завершает работу без передачи результатов работы в вызывающий сегмент.

6. Типы величин

В программе могут быть использованы пять типов постоянных и переменных величин: целые, вещественные, комплексные, логические и символьные. Тип величины должен быть определен до ее непосредственного использования.

Величины целого типа – целые числа, а также нуль. Некоторая неразбериха существует со способами визуального различия символа (буквы) «о» и нуля «0» при рукописном вводе текста программы. В отечественных руководствах по языку рекомендуется перечеркивать нуль «ø».

Величины вещественного типа – действительные числа.

Величины комплексного типа – пара действительных чисел, первое из которых определяет действительную часть числа, а второе – мнимую.

Величины логического типа (*логические величины*) – могут принимать одно из двух возможных значений: «истина» или «ложь».

Величины символьного типа – последовательности символов из основного и дополнительного наборов.

Величины различных типов по разному размещаются (представляются) в памяти электронной машины, или, иначе говоря, имеют разные *длины машинного представления*. Для хранения одного символа, входящего в состав символьной величины отводится 1 байт (8 бит) машинной памяти. Для величин целого, вещественного и комплексного типов предусмотрены две длины: *стандартная* и *нестандартная*. Под значение логической величины стандартной длины отводится 4 байта, нестандартной – 1 байт. Целая величина может иметь стандартную (4 байта) или нестандартную (2 байта) длину. Вещественная величина может иметь стандартную длину в 4 байта и двойную (нестандартную) длину в 8 байт. Отметим, что о величине вещественного типа двойной длины часто говорят как о вещественной величине *двойной точности*. В некоторых версиях компиляторов предусматривается также и нестандартная четверная длина – 16 байт. Комплексная величина занимает стандартно $2 \cdot 4 = 8$ байт, из которых 4 байта резервируется под действительную часть и 4 – под мнимую. Как и вещественные величины, комплексные могут иметь двойную и четверную длины, т. е. представляться парой действительных величин двойной или четверной длины.

При записи целой величины один бит резервируется под его знак (0 – положительное число, 1 – отрицательное), а все остальные – под его абсолютную величину. Для хранения абсолютной величины целого числа применяется так называемый *дополнительный код*. Дополнительный код положительного числа совпадает с двоичной записью самого числа, а код отрицательного числа образуется путем замены в двоичной записи абсолютной величины всех цифр на противоположные с последующим добавлением к младшему разряду единицы. Так, при нестандартной записи (2 байта = 16 бит) числа 1_{10} 15-му (старшему) биту будет присвоено значение 0, а прочим – значения 000000000000001, или

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

При том же типе записи число -1_{10} будет иметь вид

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

При таком способе хранения чисел целое число стандартной длины может принимать значения в диапазоне от $-2^{31}-1$ до $+2^{31}-1$

(−2147483648 ÷ 2147483647), а нестандартной длины от $-2^{15}-1$ до $+2^{15}-1$ (−32768 ÷ 32767).

Способ машинного хранения вещественных величин регламентируется стандартом IEEE-754, периодически обновляемым с 1985 г. Этот способ предусматривает запись действительных чисел в нормализованной форме: $M \cdot 10^p$, где M – мантисса числа ($0.1 \leqslant |M| \leqslant 1.0$), p – его порядок. При стандартной длине записи старший бит отдается под знак числа, следующие 8 бит – под порядок, а оставшиеся 23 – под мантиссу. При нестандартной – 1 бит знаковый, 11 – используются для записи порядка числа, а 52 – для его мантиссы. Чтобы не использовать дополнительный код, порядок числа записывается в смещенной форме, т. е. к его величине всегда добавляют 2^k-1, где k – количество бит (разрядов), отданных под его хранение. Использование данного способа позволяет оперировать действительными числами в диапазоне значений $10^{-39}-10^{38}$ с 11–12 значащими цифрами в мантиссе при стандартной записи и $10^{-324}-10^{308}$ с 15–16 значащими цифрами при нестандартной.

7. Константы

Константы применяются в программе для представления объектов с постоянным значением (чисел, символьных последовательностей, логических выражений и т. д.). В языке предусмотрено пять типов констант: целые, вещественные, комплексные, логические и символьные. По написанию константы можно определить ее тип и значение.

Целая константа есть последовательность десятичных цифр, начинающаяся со знака «+» или «−», и используемая для представления целого десятичного числа. Если константа обозначает положительное число, то знак «+» можно опустить. Целая константа считается величиной целого типа стандартной длины (4 байта). Например, целые константы +641, 641, 000641 обозначают одно и то же целое число 641_{10}.

Вещественные константы используются для представления действительных чисел. Они могут иметь стандартную или нестандартную (двойную) длину. Соответственно, константы стандартной

длины могут записываться как в безэкспоненциальной (тип F), так и в экспоненциальной (тип E) формах. Константы двойной длины записываются только в экспоненциальной форме (тип D).

Константа типа F начинается со знака «+» или «−» и представляет собой две последовательности десятичных цифр, отвечающие целой и дробной частям действительного числа, разделенные символом «.» (точкой). По аналогии с целыми константами, у неотрицательной вещественной константы может быть опущен знак «+». Например, константы −0.31562 и −.31562 обозначают число −0,31562. Константа типа E представляет совокупность вещественной константы типа F (*мантиссы*) и целой константы (*порядка*), разделенных символом E. Порядок константы должен содержать не более двух цифр. Например, константы −0.31562E + 00, −.31562E + 00, −0.0031562E + 02 обозначают одно и то же число −0,31562. Константы типа D имеют такую же структуру записи, как и константы типа E, но в качестве разделителя мантиссы и порядка используется символ D.

Комплексные константы состоят из пары вещественных или целых констант, разделенных запятой и помещенных в круглые скобки. Например, константа (2.31E − 01, 0.44E + 00) обозначает число 0,231 + 0,44*i*.

Логические константы могут иметь значения «истина» или «ложь», записываемые как последовательности символов «.TRUE.» или «.FALSE.».

Символьные константы состоят из последовательностей символов языка, заключенных в апострофы. Например, записи 'Fortran' и 'FORTRAN' являются символьными константами.

8. Переменные и массивы

Под *переменной* (*простой переменной*) подразумевается объект программы, принимающий в ходе выполнения программы различные значения и обозначаемый символическим именем (идентификатором). Кроме простых переменных, могут быть использованы и *переменные с индексами* (*индексируемые переменные*), также обозначаемые символьными именами. В общем случае индексируемые переменные имеют вид

$a\,(i_1, i_2, i_3, \ldots, i_n)$,

где a – символическое имя перемеренной (идентификатор), $i_k\,(k = 1, 2, .., n)$ – индекс, являющийся арифметическим выражением целого или вещественного типа. При использовании вещественного выражения в качестве индекса его значение округляется до ближайшего целого числа. Упорядоченная совокупность индексируемых переменных с одним и тем же символическим именем и всеми допустимыми значениями индексов (*ik*) называется *массивом*. Соответственно, об индексируемых переменных можно говорить как об элементах массива с тем же символическим именем.

Переменные и массивы могут относиться к целому, вещественному, комплексному, логическому и символьному типам и могут иметь стандартную или нестандартную длину. Индексируемые переменные имеют тот же тип и длину, что и массивы, элементами которых они являются. Например, список переменных XCIN, Y33 (1), Y33 (2), Y33 (3), F (1,1), F (2,1), F (1,2), F (2,2) содержит одну простую переменную с символическим именем XCIN и два списка элементов массивов Y33 и F. Первый из этих массивов содержит как минимум три элемента, а второй – четыре. По количеству индексов Y33 – одномерный массив, F – двумерный.

9. Указатель функции

Под *указателем функции* в языке FORTRAN подразумевается средство обращения к алгоритму вычисления функции вида

$f\,(x_1, x_2, \ldots, x_n)$,

где f – символическое имя функции, $x_k\,(k = 1, 2, .., n)$ – ее аргументы, которые принято называть *фактическими параметрами*. В качестве фактических параметров могут выступать константы и переменные, а также символические имена массивов, подпрограмм-функций и подпрограмм-процедур. Указатель функции должен иметь явно определенный тип и длину, по аналогии с переменной и константой.

Наличие указателя функции в тексте программы является обращением к подпрограмме, вычисляющей значение этой функции. После завершения вычислений результат присваивается данному указателю. В языке FORTRAN функцию можно вычислить двумя способами:

вызвать встроенную подпрограмму (*встроенную функцию*), а в случае ее отсутствия реализовать расчет как подпрограмму-функцию и обратиться к ней. Список встроенных функций языка достаточно обширен; он включает как часто используемые элементарные функции, так и процедуры, преобразующие типы и длины фактических параметров, и т. д. Краткий перечень встроенных функций приведен в табл. 2.

Таблица 2

Краткий перечень встроенных функций. Для указания типа и длины операнда использованы следующие обозначения: целые числа стандартной длины – I4; вещественные числа стандартной длины – R4, нестандартной – R8; комплексные числа нестандартной длины – C8

Функция	Указатель функции	Аргумент	Результат
$y = \ln x$	ALOG (X)	R4	R4
	DLOG (X)	R8	R8
	CLOG (X)	C8	C8
$y = \lg x$	ALOG10 (X)	R4	R4
	DLOG10 (X)	R8	R8
$y = \exp(x)$	EXP (X)	R4	R4
	DEXP (X)	R8	R8
	CEXP (X)	C8	C8
$y = \sqrt{x}$	SQRT (X)	R4	R4
	DSQRT (X)	R8	R8
$y = \sin x$	CSQRT (X)	C8	C8
	SIN (X)	R4	R4
	DSIN (X)	R8	R8
	CSIN (X)	C8	C8
$y = \cos x$	COS (X)	R4	R4
	DCOS (X)	R8	R8
$y = \mathrm{tg}\, x$	CCOS (X)	C8	C8
	TAN (X)	R4	R4
$y = \lvert x \rvert$	DTAN (X)	R8	R8
	IABS (X)	I4	I4

Окончание табл. 2

Функция	Указатель функции	Аргумент	Результат
	ABS (X)	R4	R4
	DABS (X)	R8	R8
	CABS (X)	C8	R4
$y = \max(x_1, \ldots, x_n)$	MAX0 (X1, ..., XN)	I4	I4
	AMAX0 (X1, ..., XN)	I4	R4
	MAX1 (X1, ..., XN)	R4	I4
	AMAX1 (X1, ..., XN)	R4	R4
	DMAX1 (X1, ..., XN)	R8	R8
$y = \min(x_1, \ldots, x_n)$	MIN0 (X1, ..., XN)	I4	I4
	AMIN0 (X1, ..., XN)	I4	R4
	MIN1 (X1, ..., XN)	R4	I4
	AMIN1 (X1, ..., XN)	R4	R4
	DMIN1 (X1, ..., XN)	R8	R8

10. Арифметические выражения

Арифметические выражения языка конструируются из операндов и знаков арифметических операций. В качестве операндов могут использоваться константы без знака, переменные (их символические имена) и указатели функций целого вещественного и комплексного типов. Любое из арифметических выражений можно заключить в круглые скобки и использовать в качестве операнда в другом арифметическом выражении. Арифметические операции обозначаются знаками: «+» (сложение), «–» (вычитание), «*» (умножение), «/» (деление) и «**» (возведение в степень). Для вычисления сложных арифметических выражений установлены следующие правила старшинства операций: 1-й уровень (первая очередь) – вычисление указателя функции, 2-й уровень (вторая очередь) – возведение в степень, 3-й уровень (третья очередь) – умножение и деление и 4-й уровень (вычисляются

в четвертую очередь) – сложение и вычитание. Если выражение содержит несколько операций одного уровня, то возведение в степень соединяет операнды справа налево, а прочие действия – слева направо. Выражения в скобках вычисляются до использования их значений в качестве операнда.

Таблица 3

Тип и длина результата операции A*x*B. Здесь *x* – операция сложения, вычитания, умножения и деления. Для указания типа и длины операнда использованы следующие обозначения: целые числа нестандартной длины – I2, стандартной – I4; вещественные числа стандартной длины – R4, нестандартной – R8; комплексные числа стандартной длины – C8, нестандартной – C16

A*x*B	I2	I4	R4	R8	C8	C16
I2	I2	I4	R4	R8	C8	C16
I4	I4	I4	R4	R8	C8	C16
R4	R4	R4	R4	R8	C8	C16
R8	R8	R8	R8	R8	запрещено	C16
C8	C8	C8	C8	запрещено	C8	C16
C16	C16	C16	C16	C16	C16	C16

Таблица 4

Тип и длина результата операции возведения в степень AB**

A**B	I2	I4	R4	R8
I2	I2	I4	R4	R8
I4	I4	I4	R4	R8
R4	R4	R4	R4	R8
R8	R8	R8	R8	R8
C8	C8	C8	C8	запрещено
C16	C16	C16	запрещено	запрещено

Значение арифметического выражения может относиться к целому, вещественному и комплексному типу и иметь как стандартную, так и нестандартную длину. Правила определения типа и длины значений арифметических операций в зависимости от типа и длины операндов приведены в табл. 3, 4.

11. Отношения

Отношение является отдельным типом логического выражения, в котором операндами являются арифметические выражения целого или вещественного типа, а результатом – логические значения «истина» или «ложь». Отношение имеет вид:

A*x*B,

где операнды A и B могут различаться как типами, так и длиной, а *x* – знак операции отношения. В процессе вычисления результата отношения операнды преобразуются к одной длине и одному типу, как и операнды в арифметических выражениях. Знаками операций отношения являются следующие последовательности основных символов:

.LT. (меньше)
.LE. (меньше или равно)
.EQ. (равно)
.NE. (не равно)
.GT. (больше)
.GE. (больше или равно)

Результат операции отношения принимает значение «истина», если отношение удовлетворяется для входящих в него арифметических выражений, и «ложь» – в противном случае.

12. Логические выражения

Логические выражения образуются из операндов логического типа и логических операций. Как и в случае арифметических выражений, логические операнды могут иметь сложную структуру и включать в себя логические константы, переменные, указатели

функций и отношения. При необходимости в логических выражениях могут быть использованы круглые скобки, и значения выражений в скобках будут вычислены до их использования в качестве операндов. К логическим операциям относятся: .NOT. – отрицание (одноместная операция), .AND. – логическое умножение (двуместная операция) и .OR. – логическое сложение (двуместная операция). Правила выполнения логических операций приведены в табл. 5.

Таблица 5

Правила выполнения логических операций

A	.TRUE.	.TRUE.	.FALSE.	.FALSE.
B	.TRUE.	.FALSE.	.TRUE.	.FALSE.
.NOT.A	.FALSE.	.FALSE.	.TRUE.	.TRUE.
A. AND.B	.TRUE.	.FALSE.	.FALSE.	.FALSE.
A.OR.B	.TRUE.	.TRUE.	.TRUE.	.FALSE.

Как и в арифметических выражениях, логические операнды могут иметь различные длины. Правила определения длины результата для операций с операндами различной длины приведены в табл. 6.

Таблица 6

Правила определения длины результата логических операций. Обозначения L1 и L4 используются для логических операндов нестандартной (1 байт) и стандартной (4 байта) длины

Первый операнд	Второй операнд	Операция	Результат
–	L4	.NOT.	L4
–	L1	.NOT.	L1
L4	L4	.AND.,.OR.	L4
L4	L1	.AND.,.OR.	L4
L1	L4	.AND.,.OR.	L4
L1	L1	.AND.,.OR.	L1

Вычисление значения логического выражения проводится с последовательным выполнением операций слева направо с учетом старшинства операций и скобок. Установлены следующие правила старшинства:

1-й уровень – вычисление арифметических выражений;
2-й уровень – вычисление отношений;
3-й уровень – отрицание;
4-й уровень – логическое умножение;
5-й уровень – логическое сложение.

13. Символьные выражения

Символьное выражение определяет последовательность символов, которая собственно и является его значением. Возможны два способа обращения к символьной последовательности: обращение ко всей последовательности символов (переменной) и *обращение к ее подпоследовательности*. Обращение к подпоследовательности осуществляется конструкцией вида ***name*** (***min: max***), где ***name*** – символическое имя символьной переменной, ***min, max*** – индексы начального и конечного символов, отбираемых в подпоследовательность. Например, символьная переменная STRING имеет длину в 6 байт и значение 'ASTON1'. Тогда подпоследовательность STRING (3:4) будет иметь значение 'TO', STRING (3:) – 'TON1', STRING (:3) – 'AST', и STRING (:) – 'ASTON1'.

Простейшим символьным выражением может быть символьная константа, переменная или обращение к функции. Из простейших символьных выражений можно выстроить более сложные выражения, используя операцию *конкатенации* (сшивки) символьных последовательностей. Сшивка задается символами «//». Например, имеется две символьные переменные: A (2 байта) и B (1 байт). Первая из них содержит последовательность символов 'ye', а вторая – символ 'S'. Тогда, сшивая значения этих переменных операцией A//B, получим новую последовательность 'yeS' длиной 3 байта. Сшивая в обратном порядке: B//A, получим последовательность 'Sye'.

14. Операторы присваивания

Операторы присваивания делятся на операторы, присваивающие переменным арифметические, логические или символьные значения, и операторы, присваивающие значения метки.

Арифметический, логический и символьный операторы присваивания можно представить в общем виде

n variable = value,

где ***n*** – метка оператора (или пробел), ***variable*** – имя простой переменной или элемента массива, а ***value*** – соответствующего типа выражение. Выполнение оператора присваивания включает в себя вычисление выражения ***value*** с последующим присваиванием вычисленного значения переменной ***variable***. Если типы или длины ***variable*** и ***value*** не совпадают, то в процессе выполнения оператора будет осуществлено преобразование типа и длины значения ***value*** к типу и длине переменной ***variable***.

Присваивание значения метки осуществляется посредством оператора вида:

n ASSIGN ***m*** TO ***name,***

где ***n*** – метка оператора (или пробел), ***m*** – целая положительная константа без знака, значение которой совпадает с одной из используемых в программе меток, а ***name*** – целая переменная стандартной длины. Переменная, получившая значение метки, далее должна использоваться в операторе *перехода по предписанию*.

Например, в результате выполнения оператора

ASSIGN 22 TO LABEL

целой переменной стандартного типа LABEL будет присвоено значение 22.

15. Оператор безусловного перехода

Оператор безусловного перехода имеет вид:

n GO TO ***label,***

где ***n*** – метка оператора (или пробел), ***label*** – метка некоторого оператора, находящегося внутри данной программы (подпрограммы). Действие оператора сводится к передаче управления программой оператору с меткой ***label***.

16. Оператор перехода по предписанию

Оператор перехода по предписанию имеет вид:
n GO TO ***label,*** (***n***$_1$***, n***$_2$***, …, n***$_k$),
где ***n*** – метка оператора (или пробел), ***label*** – простая переменная целого типа стандартной длины, ***n***$_j$ (j = 1, 2, …, k) – метки некоторых операторов, находящихся внутри данной программы (подпрограммы). К моменту выполнения этого оператора значение переменной ***label*** должно быть уже определено (см. раздел «Операторы присваивания») и должно совпадать с одной из меток списка ***n***$_j$ (j = 1, 2, …, k). Если какое-либо из перечисленных требований не выполнено, то действие оператора перехода по предписанию не определено.

17. Оператор перехода по вычислению

Оператор перехода по вычислению имеет вид:
n GO TO (***n***$_1$***, n***$_2$***, …, n***$_k$) ***variable***,
где ***n*** – метка оператора (или пробел), ***variable*** – арифметическое выражение целого типа стандартной или нестандартной длины, ***n***$_j$ (j = 1, 2, …, k) – метки некоторых операторов, находящихся внутри данной программы (подпрограммы). Если значение выражения ***variable*** находится в диапазоне от 1 до k, то оператор перехода осуществляет переход по метке ***n***$_j$, идущей в списке под номером, совпадающим со значением ***variable***. Если значение ***variable*** выходит за границы диапазона (***variable***<1 или ***variable***>k), то выполняется следующий за оператором перехода оператор. На момент выполнения этого оператора значение выражения ***variable*** должно быть определено.

18. Операторы условного перехода

К операторам условного перехода относятся три оператора: *условный арифметический оператор*, *условный логический оператор* и *блочный условный оператор*. Все эти операторы относятся к классу исполняемых операторов.

Условный арифметический оператор имеет вид:

n IF (***variable***) $\boldsymbol{n}_1, \boldsymbol{n}_2, \boldsymbol{n}_3$,

где ***n*** – метка оператора (или пробел), ***variable*** – арифметическое выражение целого или вещественного типа стандартной или нестандартной длины, $\boldsymbol{n}_j$ (j = 1, 2, 3) – метка некоторых операторов, находящихся внутри данной программы (подпрограммы). При выполнении данного оператора сначала вычисляется значение арифметического выражения ***variable***, после чего управление передается одному из операторов с метками $\boldsymbol{n}_1$, $\boldsymbol{n}_2$, $\boldsymbol{n}_3$, если значение ***variable*** меньше нуля ($\boldsymbol{n}_1$), равно нулю ($\boldsymbol{n}_2$) или больше нуля ($\boldsymbol{n}_3$).

Условный логический оператор имеет вид:

n IF (***variable***) ***operator***,

где ***n*** – метка оператора (или пробел), ***variable*** – логическое выражение стандартной или нестандартной длины, ***operator*** – выполняемый оператор, отличный от логического условного оператора, блочного условного оператора и оператора цикла (см. раздел «Оператор цикла»). Выполнение оператора состоит из вычисления логического выражения ***variable*** с последующим выполнением оператора ***operator***, если значение ***variable*** истинно. В противном случае (значение ***variable*** ложно) выполняется следующий по порядку оператор.

Блочный условный оператор имеет вид:

n IF (***variable***) THEN
Block0
ELSE
Block1
END IF

где ***n*** – метка (или пробел), ***variable*** – логическое выражение стандартной или нестандартной длины, ***Block0*** и ***Block1*** – последовательности выполняемых операторов. Если значение ***variable*** – истинно, то выполняется последовательность ***Block0***, в противном случае – ***Block1***. Блочный условный оператор может быть упрощен путем удаления условия ELSE и последовательности ***Block1***. В таком случае, если ***variable*** – ложно, то выполняется следующий после условного оператор.

Существует и еще одна разновидность блочного условного оператора более общего вида:

n IF ($\boldsymbol{variable}_0$) THEN
$\boldsymbol{Block}_0$

ELSE IF (***variable***$_1$) THEN
Block$_1$
ELSE IF (***variable***$_2$) THEN
Block$_2$
…… …..
ELSE IF (***variable***$_k$) THEN
Block$_k$
END IF

Выполнение этой конструкции начинается с проверки значения логического выражения ***variable***$_0$, и если оно истинно, то выполняется последовательность операторов ***Block***$_0$. В противном случае начинается выполнение условного оператора

IF (***variable***$_1$) THEN
Block$_1$
ELSE IF (***variable***$_2$) THEN
Block$_2$
…… …..
ELSE IF (***variable***$_k$) THEN
Block$_k$
END IF
и т. д.

19. Вспомогательные операторы управления

К этому типу операторов относятся операторы *продолжения*, *паузы*, *возврата* и *останова*. Операторы возврата и останова уже были рассмотрены выше в разделе «Структура программы», описывать их повторно нет необходимости.

Оператор *продолжения* имеет вид:

n CONTINUE,

где ***n*** – метка (или пробел). Этот оператор относится к классу *выполняемых*, но не выполняет никаких действий и используется, как правило, для передачи управления или завершения оператора цикла.

Оператор *паузы* имеет вид:

n PAUSE ***m***,

где ***n*** – метка (или пробел), ***m*** – целая константа, содержащая от 1 до 5 десятичных цифр, или символьная константа. Выполнение

этого оператора должно приводить к временной остановке исполнения программы с передачей операционной системе значения параметра ***m***. В дальнейшем выполнение программы может быть продолжено по указанию пользователя. В многозадачных операционных системах выполнение этого оператора, скорее всего, приведет к выводу на управляющий терминал значения ***m*** без остановки исполнения программы.

20. Оператор цикла

Оператор цикла имеет структуру вида

n DO ***label, variable*** = m_1, m_2, m_3
Block
label continue,

где ***n*** – метка (или пробел), ***label*** – метка конца цикла, ***variable*** – параметр цикла (переменная целого типа стандартной или нестандартной длины), m_1 – начальное значение параметра цикла, m_2 – конечное значение параметра цикла, m_3 – приращение параметра цикла, ***Block*** – последовательность выполняемых операторов, ***continue*** – последний оператор цикла. Для удобства за составляющими цикла закреплены следующие названия: первую строку цикла (***n*** DO ***label, variable*** = m_1, m_2, m_3) принято называть *заголовком цикла* (иногда также называют конструкцию ***variable*** = m_1, m_2, m_3), последовательность исполняемых операторов внутри цикла (***Block***) – *телом цикла*, а последний оператор (***label continue***) – *конечным оператором цикла*. Первый оператор из последовательности ***Block*** также имеет свое название – *начальный оператор цикла*. Заголовок цикла может иметь и более простой вид; могут быть опущены, во-первых, запятая между параметрами ***label*** и ***variable***, а во-вторых, – параметр m_3, если его значение равно единице. В роли параметров цикла (m_1, m_2, m_3) могут выступать простые переменные или константы целого типа. Конечный оператор цикла должен быть выполняемым оператором, отличным от операторов RETURN, STOP, END, GOTO и условного арифметического оператора. Конечный оператор цикла может совпадать с начальным оператором; например, конструкция

```
DO 10 I = 1,5
```

10 X = X + Y (I)

является корректной. Зачастую удобно в качестве конечного оператора использовать оператор CONTINUE (см. раздел «Вспомогательные операторы управления»).

Выполнение оператора цикла начинается с присвоения ***variable*** = m_1, затем выполняется тело цикла. После выполнения конечного оператора переменной ***variable*** присваивается значение ***variable*** = $m_1 + m_3$. Если после этого значение ***variable*** не превосходит значения m_2, то тело цикла выполняется повторно, в противном случае управление программой переходит к оператору, следующему за конечным оператором цикла. После выхода из цикла по условию окончания цикла значение параметра цикла становится неопределенным.

21. Описание типов переменных и массивов

Типы используемых в программе (подпрограмме) переменных и массивов могут быть заданы несколькими способами. Тип переменной величины можно определить явно, воспользовавшись одним из *операторов описания типа*. Если операторы описания типа охватывают не все используемые в программе переменные, то типы оставшихся определяются по умолчанию. В этом случае величины, чьи символические имена начинаются с букв I, J, K, L, M, N, предполагаются целыми стандартной длины; если символическое имя величины начинается с любой другой буквы, то она считается вещественной величиной стандартной длины.

В том случае, когда описываемые переменные (массивы) относятся к целому, логическому, вещественному или комплексному типу, *явный оператор типа* имеет вид:

type length name$_1$***length***$_1$ (***size***$_1$)/***value***$_1$/, …, ***name***$_n$***length***$_n$ (***size***$_n$)/***value***$_n$/

где ***type*** – указатель типа величины, ***lenght*** – общий указатель длины. Далее следует список величин ***name***$_j$***length***$_j$ (***size***$_j$)/***value***$_j$/ (j = 1, 2, …, *n*), указывающий: ***name***$_j$ – символическое имя величины, ***length***$_j$ – ее длину, ***size***$_j$ – размерность и ***value***$_j$ – начальное значение.

Для указания типа величин ***name***$_j$ из списка используются служебные слова INTEGER (целый тип), REAL (вещественный тип), COMPLEX (комплексный тип) и LOGICAL (логический тип).

Указатель длины состоит из символа «*», за которым следует целое число без знака, указывающее длину величины в байтах.

Размер (указатель размерности) $\boldsymbol{size}_j$ задается набором *граничных пар* ($\boldsymbol{min}_1$: $\boldsymbol{max}_1$, $\boldsymbol{min}_2$: $\boldsymbol{max}_2$, …, $\boldsymbol{min}_k$, $\boldsymbol{max}_k$), где $k \leqslant 7$ – размерность массива, $\boldsymbol{min}_j$ и $\boldsymbol{max}_j$ ($j = 1, 2, \ldots, k$) – константы или выражения целого типа. *Нижний элемент* граничной пары ($\boldsymbol{min}_j$) всегда должен иметь значение, не превышающее значение *верхнего элемента* ($\boldsymbol{max}_j$). Нижний элемент может быть опущен вместе с последующим двоеточием, если его значение равно единице.

По способу задания размера в языке FORTRAN существуют три типа массивов: массив *постоянного размера*, массив *регулируемого размера* и массив *предполагаемого размера*. У первого типа массивов (постоянного размера) значения граничных пар задаются константами, у второго – переменными или выражениями целого типа, значения которых определены до описания самого массива, у третьего – верхний элемент последней граничной пары ($\boldsymbol{max}_k$) – определяется символом «*». Отметим, что массивы регулируемого или предполагаемого размера могут использоваться только в списках формальных параметров подпрограмм. Составляющие массив элементы хранятся (в памяти машины) таким образом, что если для двух элементов с индексами (…, $i_1, j_1, j_2, \ldots, j_k$) и (…, $i_2, j_1, j_2, \ldots, j_k$) выполняется условие $i_2 > i_1$, то элемент (…, $i_2, j_1, j_2, \ldots, j_k$) располагается в памяти дальше от начала массива (первого элемента), чем элемент (…, $i_1, j_1, j_2, \ldots, j_k$). Данный порядок размещения элементов принято называть *столбцовым*, поскольку его применение к двумерному массиву приводит к тому, что сначала в память записываются элементы первого столбца, далее – второго, третьего, и т. д.

Начальное значение ($\boldsymbol{value}_j$) переменной является константой соответствующего типа, массива – набором констант соответствующего типа, разделенных запятыми, и/или *констант с повторителем*. Под константой с повторителем подразумевается выражение вида ***rep*value***, где ***rep*** – целое число без знака (повторитель), а ***value*** – повторяемая константа. Начальные значения присваиваются элементам массива в порядке расположения элементов в памяти, и если число начальных значений в списке меньше числа элементов, то начальные значения присваиваются только первым элементам массива.

Присутствие в явном операторе описания типа всех перечисленных составляющих не обязательно; при необходимости можно

не применять указатели длины и размерности, а также не присваивать элементам начальные значения.

В качестве примера рассмотрим конструкцию

REAL X1, X2*8, Y1 (1:2,–1:2), Y2 (2,3)*8/3*1.0,1*1.4142/

Первое, что можно сказать о данной конструкции – она определяет две простые вещественные переменные (X1 и X2) и два двумерных вещественных массива (Y1 и Y2). За исключением X2, имеющей нестандартную длину (8 байт), остальным переменным определена стандартная длина (4 байта). Четырем первым элементам массива Y2 присвоены начальные значения: Y2 (1,1), Y2 (1,2), Y2 (1,3) присваивается значение 1.0, а Y2 (2,1) – значение 1.4142. Двум оставшимся элементам: Y2 (2,2) и Y2 (2,3) начальные значения не присваиваются.

Символьные переменные тоже могут быть описаны явно, с помощью оператора вида:

CHARACTER ***lenght name***$_1$***lenght***$_1$ (***size***$_1$), …, ***name***$_n$***lenght***$_n$ (***size***$_n$),

где ***lenght*** – общий указатель длины (число или выражение целого типа). Этот параметр опускается, если все описываемые в операторе переменные имеют длину в 1 байт. Формальные параметры подпрограмм могут быть описаны как символьные переменные предполагаемой длины посредством указателя длины вида «* (*)». В этом случае длина символьной переменной будет такой же, как и длина фактического параметра. Параметры ***name***$_j$***lenght***$_j$ (***size***$_j$) ($j = 1, 2, \ldots, n$), входящие в список, указывают: ***name***$_j$ – символическое имя величины, ***lenght***$_j$ – ее длину (число или выражение целого типа), ***size***$_j$ – размерность (для массива).

Второй способ описания переменных (массивов) заключается в использовании комбинации из *неявного оператора типа* и *оператора размера* (размеров). Первый оператор имеет вид:

IMPLICIT ***type***$_1$***lenght***$_1$***list***$_1$, …, ***type***$_n$***lenght***$_n$***list***$_n$

где ***type***$_j$ ($j = 1, 2, \ldots, n$) – указатель типа величины, ***lenght***$_j$ – указатель длины, ***list***$_j$ – список указателей величин. Список указателей величин определяет величины перечислением их первых букв через запятую или через дефис. Неявный оператор типа должен предшествовать всем операторам описания типа, но следовать за операторами присвоения начальных значений PARAMETER. В том случае когда тип некоторой величины указан как явно, так и неявно, предпочтение отдается явному способу описания.

Например, оператор IMPLICIT REAL *8 (A – H, O, Z) указывает на то, что все величины, чьи символические имена начинаются с букв A – H, O, Z относятся к переменным вещественного типа нестандартной (двойной) длины.

Оператор размера имеет вид

DIMENSION $\boldsymbol{name}_1$ ($\boldsymbol{size}_1$), …, $\boldsymbol{name}_n$ ($\boldsymbol{size}_n$),

где $\boldsymbol{name}_j$ ($j = 1, 2, \ldots, n$)– символическое имя массива, $\boldsymbol{size}_j$ – указатель размерности, задаваемый набором *граничных пар* ($\boldsymbol{min}_1$: $\boldsymbol{max}_1$, $\boldsymbol{min}_2$: $\boldsymbol{max}_2$, …, $\boldsymbol{min}_k$, $\boldsymbol{max}_k$), где $k \leqslant 7$ – размерность массива, $\boldsymbol{min}_j$ и $\boldsymbol{max}_j$ ($j = 1, 2, \ldots, k$) – константы или выражения целого типа. *Нижний элемент* граничной пары ($\boldsymbol{min}_j$) всегда должен иметь значение, не превышающее значения *верхнего элемента* ($\boldsymbol{max}_j$). Нижний элемент может быть опущен вместе с последующим двоеточием, если его значение равно единице.

22. Начальные значения переменных величин

В ряде случаев необходимо присвоить известные значения некоторым переменным величинам в самом начале выполнения программы. Данные действия можно рассматривать и как присвоение начального значения некоторой переменной величине, и, как присвоение постоянной величине (и часто используемой) символического имени. Для присвоения начальных в языке имеется оператор DATA, а для присвоения символических имен – оператор PARAMETER.

Оператор DATA имеет вид:

DATA $\boldsymbol{name}_1/\boldsymbol{value}_1/$, …, $\boldsymbol{name}_n/\boldsymbol{value}_n/$,

где $\boldsymbol{name}_j$ – список символических имен переменных величин, $\boldsymbol{value}_j$ – набор констант, разделенных запятыми, и/или констант с повторителем, совпадающих по типам и длинам с переменными из списка $\boldsymbol{name}_j$.

Оператор PARAMETER имеет вид:

PARAMETER ($\boldsymbol{name}_1 = \boldsymbol{value}_1$, $\boldsymbol{name}_2 = \boldsymbol{value}_2$, …, $\boldsymbol{name}_n = \boldsymbol{value}_n$),

где $\boldsymbol{name}_j$ – символическое имя переменной величины, $\boldsymbol{value}_j$ – выражение или константа, совпадающая по типу и длине с переменной. Присвоенное символическое имя можно использовать в определении других символических имен, в выражениях и т. д.

23. Оператор общих областей

Язык FORTRAN позволяет размещать величины, используемые в разных программных модулях (программе и подпрограммах), в одних и тех же областях оперативной памяти (общих областях). Для создания таких областей используется *оператор общих областей*, имеющий вид:

COMMON/$\boldsymbol{name}_1$/$\boldsymbol{list}_1$,/$\boldsymbol{name}_2$/$\boldsymbol{list}_2$, …,/$\boldsymbol{name}_n$/$\boldsymbol{list}_n$,

где $\boldsymbol{name}_j$ – символическое имя (идентификатор) общей области, $\boldsymbol{list}_j$ – список величин, помещаемых в общую область. Список величин помещаемых в общую область представляет собой набор символических имен простых переменных, массивов без указания размерности (если массив уже описан в предыдущих строках программы) или массивов с указанием размерности (если он не был ранее указан в операторе размеров или в явном операторе типа). Общие области подразделяются на *именованные* (имеющие символическое имя) и *неименованные* (не имеющие символического имени). В одном программном модуле может быть описано несколько именованных областей и единственная неименованная область.

При использовании общих областей должны соблюдаться следующие условия: а) размер одноименной общей области памяти должен быть одинаковым во всех программных модулях; б) если общая область вводится для обмена данными между программными модулями, то типы и длины данных в программных модулях должны совпадать.

24. Оператор эквивалентности

Оператор эквивалентности имеет вид:

EQUIVALENCE ($\boldsymbol{a}_1$, $\boldsymbol{a}_2$, …, $\boldsymbol{a}_k$), ($\boldsymbol{b}_1$, $\boldsymbol{b}_2$, …, $\boldsymbol{b}_n$), …, ($\boldsymbol{z}_1$, $\boldsymbol{z}_2$, …, $\boldsymbol{z}_r$),

где в круглые скобки помещаются группы эквивалентности, включающие простые переменные или элементы массива. Учитывая, что элементы массивов хранятся в последовательном порядке, совмещение отдельных необходимых элементов будет приводить к частичному совмещению других элементов этих массивов.

25. Организация ввода-вывода данных

Для хранения данных в вычислительной машине может быть использовано множество запоминающих устройств (жесткие диски, магнитные ленты и т.п.), различающихся как по способам физического хранения информации, так и по скорости доступа к ней. Данные размещаются на запоминающих устройствах в виде *наборов данных* (в современной терминологии – *файлов*). На физическом уровне (уровне устройства) доступ к файлам происходит посредством *физических записей*, каждая из которых имеет в составе определенное количество *позиций*, называемое *длиной физической записи*. Под *позицией* здесь подразумевается элемент физического носителя информации? обеспечивающий хранение элементарной порции данных. Элементарными порциями могут быть отдельные символы, восьми- и шестнадцатиразрядные коды и т. д. Чтение и запись данных программой из файлов осуществляется порциями, которые принято называть *логическими записями*. Зависимость между физической и логической длинами записей файла определяется форматом его физических записей и способом его организации.

В языке FORTRAN предполагается, что все используемые программой данные хранятся в файлах, размещаемых на запоминающих устройствах машины или в оперативной памяти. При этом все этапы управления запоминающими устройствами осуществляются операционной системой. По способу размещения файлы подразделяют на *внешние* и *внутренние*. Внешний файл – это совокупность данных на запоминающих устройствах, а также поток данных, выводимых на печатающее устройство или экран и вводимых с клавиатуры. Внутренний файл – это символьная переменная или элемент символьного массива. Любой файл снабжается уникальным именем. Для внутреннего файла имя – это символическое имя символьной переменной или элемента символьного массива. Для внешнего файла имя – это набор символов, удовлетворяющий правилам именования файлов в операционной системе. Доступ к файлам осуществляется посредством группы операторов, называемых *операторами ввода-вывода*.

Операторы ввода-вывода обращаются к файлу по *идентификатору устройства*. Для внутреннего файла идентификатор устройства – это символическое имя внутреннего файла. Для внешнего

файла идентификатор устройства задается константой или выражением целого типа, а также символом «*». Значение идентификатора должно находиться в диапазоне от нуля до 32 767. Перед обращением к внешнему файлу с помощью операторов ввода-вывода, должна быть установлена связь между устройством и файлом. Процесс установления связи называется *открытием* (*присоединением*) внешнего файла. Файл может быть присоединен к устройству *явно*, *неявно* и *предварительно*. При явном присоединении файл остается присоединенным до отмены присоединения (*закрытия*) самой программой или до завершения ее работы. Неявно присоединенный файл закрывается автоматически по завершении программы. При предварительном присоединении внешний файл подключается к устройству по умолчанию в начале выполнения программы. Предварительно присоединяются устройства с идентификаторами * (или 0), 5, 6. Первому из них отвечают устройства, используемые для ввода-вывода данных по умолчанию. Для операционной системы UNIX это стандартные потоки ввода-вывода stdin и stdout (ввод с клавиатуры и вывод на текстовый терминал). Идентификаторы 5 и 6 соответствуют тем же устройствам. К устройствам 0, 5 и 6 могут быть присоединены явно любые внешние файлы. Для возвращения предварительного присоединения внешние файлы необходимо явно отсоединить от этих устройств. Устройство «*» всегда присоединено к клавиатурному вводу и терминалу; его нельзя использовать для явного присоединения других устройств.

Файл может быть присоединен как *входной*, *выходной* или *обновляемый*. В первом случае файл доступен только для чтения содержащихся в нем данных, во втором – только для записи, а в последнем – для чтения и записи.

Любой файл вне зависимости от типа состоит из логических записей (записей), каждая из которых является последовательностью символов (позиций). По методу доступа к записям файлы разделяют на файлы *последовательного* и *прямого доступа*. Файлы последовательного доступа допускают только последовательное обращение к записям, начиная с первой и переходя к последующей (или возвращаясь к предыдущей). Файлы прямого доступа обеспечивают обращение к нужной записи в произвольном порядке. Доступ к внутреннему файлу, а также к внешним файлам, хранящимся на устройствах последовательного доступа (клавиатура,

терминал и печатающее устройство), может осуществляться только последовательно. Файлы прямого доступа должны размещаться на устройствах прямого доступа (например, жестких дисках). К файлу, присоединенному для прямого доступа, можно обращаться и при помощи операторов ввода-вывода последовательного доступа. Записи в файле последовательного доступа могут различаться по длине, в файле прямого доступа – должны иметь одинаковую длину.

По способу представления данных записи различаются на форматные и бесформатные. Форматные записи содержат данные в символьном (внешнем) представлении, бесформатные – из данных во внутреннем (шестнадцатиричном или двоичном) представлении. При вводе данные форматной записи преобразуются из внешнего (символьного) представления во внутреннее представление в памяти, а при выводе – наоборот. Бесформатная запись содержит данные во внутреннем представлении, и обмен данными между памятью и бесформатным файлом происходит непосредственно во внутреннем представлении. Соответственно, длина бесформатной записи измеряется в байтах, а форматной – в символах.

Оператор, открывающий внешний файл имеет вид

OPEN (***list***),

где ***list*** – список спецификаций: UNIT, IOSTAT, ERR, FILE, STATUS, ACCESS, FORM, RECL, BLANK. Список должен включать, как минимум, спецификацию UNIT. Все прочие спецификации могут быть исключены из списка. Порядок следования спецификаций в списке после UNIT произволен. Спецификации имеют следующие форматы и назначение:

UNIT = ***unit,***

где ***unit*** – выражение целого типа с неотрицательным значением, соответствующим идентификатору устройства.

IOSTAT = ***iostat,***

где ***iostat*** – символическое имя простой переменной или элемента массива целого типа. При успешном выполнении оператора ***iostat*** присваивается значение 0 (нуль), в противном случае – положительное значение.

ERR = ***err,***

где ***err*** – метка оператора, который будет выполняться при аварийном (неудачном) выполнении оператора OPEN.

FILE = ***file,***

где ***file*** – символьное выражение (символьная константа), значением которого является название открываемого файла. Название файла может заканчиваться произвольным количеством пробелов.

STATUS = ***status,***

где ***status*** – символьное выражение, указывающее статус файла при его открытии с возможными значениями 'OLD', 'NEW', 'SCRATCH', 'UNKNOWN'. 'NEW' – файл создается при выполнении оператора, 'OLD' – открываемый файл уже существует, 'UNKNOWN' – может быть открыт уже существующий файл или создан новый, 'SCRATCH' – открывается временный файл. При наличии спецификации STATUS = 'SCRATCH' название файла выбирается компилятором (операционной системой), спецификация FILE = ***file*** должна отсутствовать в списке спецификаций. Отсутствие спецификации STATUS = ***status*** эквивалентно наличию спецификации STATUS = 'UNKNOWN'.

ACCESS = ***access,***

где ***access*** – символьное выражение, с возможным значением 'SEQUENTIAL', если выбран последовательный способ доступа к записям файла, или 'DIRECT' – в случае прямого доступа. Отсутствие данной спецификации в списке эквивалентно указанию спецификации ACCESS = 'SEQUENTIAL'.

FORM = ***form,***

где ***form*** – символьное выражение, принимающее значения 'FORMATTED' или 'UNFORMATTED' в зависимости от типа записи в файле (форматная или бесформатная).

RECL = ***recl,***

где ***recl*** – выражение целого типа с положительным значением, определяющее длину записи в файле. Для файла с форматными записями длина записи указывается в символах, с бесформатными – в байтах. Данная спецификация обязательна для файла с прямым доступом.

BLANK = ***blank,***

где ***blank*** – символьное выражение, принимающее значения 'NULL' или 'ZERO'. 'NULL' означает, что пробелы в числовых полях форматных записей игнорируются, 'ZERO' – пробелы трактуются как нули. По умолчанию файл открывается со спецификацией BLANK = 'NULL'.

Оператор, закрывающий (отсоединяющий) внешний файл, имеет вид:

CLOSE (***list***)

где ***list*** – список спецификаций UNIT, IOSTAT, ERR, STATUS. Обязательно наличие спецификации UNIT, прочие могут быть опущены. Порядок следования спецификаций IOSTAT, ERR, STATUS произвольный. Первые три спецификации полностью совпадают с одноименными спецификациями оператора OPEN. Последняя имеет значение

STATUS = ***status,***

где ***status*** – символьное выражение, указывающее статус файла при его отсоединении с возможными значениями 'KEEP', 'DELETE'. 'KEEP' – файл сохраняется после выполнения оператора (программы), 'DELETE' – уничтожается. При наличии спецификации STATUS = 'SCRATCH' выражение ***status*** может иметь только значение 'DELETE'. Отсутствие спецификации STATUS = ***status***, эквивалентно наличию спецификации STATUS = 'KEEP' для всех файлов, кроме временных.

Для изменения текущей позиции файла и помещения в него записи конца файла служат операторы BACKSPACE, REWIND и ENDFILE. Эти операторы имеют вид:

BACKSPACE (***unit***),
BACKSPACE (***list***),
REWIND (***unit***),
REWIND (***list***),
ENDFILE (***unit***),
ENDFILE (***list***),

где ***unit*** – идентификатор внешнего устройства (целое выражение с неотрицательным значением), ***list*** – список спецификаций UNIT, IOSTAT, ERR, совпадающих с одноименными спецификациями оператора OPEN. Перечисленные операторы выполняются только для файлов с последовательным доступом.

Оператор BACKSPACE переводит текущую позицию в файле на одну запись назад. Оператор REWIND перемещает текущую позицию в начало файла. Последний из перечисленных операторов вызывает вывод в файл записи конца файла. После его выполнения любая операция с файлом требует применения операторов BACKSPACE или REWIND, т. е. «перемотки» текущей позиции в начальное или промежуточное (отличное от конца файла) положение.

26. Последовательный ввод-вывод данных

Файл, присоединенный к программе, может быть использован для ввода или вывода данных. Последовательный форматный ввод данных осуществляется оператором READ, вывод – оператором WRITE или PRINT.

Оператор ввода имеет вид:

READ (UNIT = ***unit***, FMT = ***fmt***, IOSTAT = ***iostat***, ERR = ***err***, END = ***end***) ***list,***

где ***unit*** – идентификатор устройства (см. предыдущий раздел), ***fmt*** – спецификация формата ввода значений переменных из списка ***list***, IOSTAT, ERR, END – необязательные параметры.

Спецификации IOSTAT и ERR имеют такое же назначение, как и в операторе OPEN, спецификация ***end*** (константа целого типа с положительным значением) указывает на метку оператора, который будет выполнен по достижении конца файла. Запись оператора может быть упрощена за счет исключения ключевых слов 'UNIT = ' и 'FMT = ' с сохранением порядка следования спецификаций. В случае использования свободного формата ввода (вывода) спецификация формата в операторе заменяется символом '*'.

Вывод данных осуществляется с помощью операторов

WRITE (UNIT = ***unit***, FMT = ***fmt***, IOSTAT = ***iostat***, ERR = ***err***) ***list***

или, в выводе на стандартное устройство,

PRINT ***fmt***, ***list,***

где все компоненты имеют тот же смысл, что и в операторе READ.

Используемый в этих операторах список ***list*** определяет объекты, которым будут присвоены некоторые значения из числа существующих в открытом файле или чьи значения будут сохранены в открытом файле. Под объектами здесь следует понимать простые переменные, элементы массива, массив и символьную последовательность.

Формат передаваемых данных задается непосредственно в операторах ввода-вывода как символьная константа или специфицируется с помощью невыполняемого оператора

label FORMAT (***list***),

где ***label*** – метка оператора, ***list*** – спецификация формата. Спецификация формата включает набор *дескрипторов формата*, отвечающих за конвертацию величин в символьную строку при

выводе или во внутреннее (машинное) представление при вводе, и отделяемых друг от друга запятыми. Дескрипторы формата по назначению делятся на три типа: *дескрипторы данных*, *дескрипторы управляющие* и *дескрипторы символьных (текстовых) строк*. Дескрипторы данных и группы дескрипторов могут иметь предшествующий им *коэффициент повторения* (целая положительная константа без знака). Повторяемые группы должны быть заключены в круглые скобки.

Для целых величин зарезервирован дескриптор типа I*w*, где *w* – ширина поля. Целая величина считывается (записывается) из этого поля смещенной к его правому краю. Для вещественных величин в языке имеется два базовых дескриптора F и E, имеющих вид F*w.d* и E*w.d,* где *w* – ширина поля и *d* – количество цифр после десятичной точки. Дескриптор F представляет вещественное число как последовательность символов, отвечающих целой и дробным частям, разделенным символом «.»; дескриптор E – как значащую часть, по абсолютному значению не превышающую единицы, и поле порядка, состоящее из разделяющего символа «E», знака порядка и двух символов значения порядка. Комплексные величины можно конвертировать (преобразовывать) парами дескрипторов F или E типов в зависимости от значений их действительной и мнимой частей. Логические величины могут преобразовываться дескриптором L*w*, где *w* – ширина поля. При выводе в крайней правой позиции форматного поля должен отображаться символ «T» или «F». Символьные величины – дескриптором A*w*. Кроме указанных существуют и другие типы дескрипторов данных (I*w.m*, E*w.d*E*e*, ES*w.d*, EN*w.d* и т. д.); для их изучения можно обратиться к работам [1–3].

Символьную строку (символьную константу) можно включить в спецификацию формата непосредственно как в оператор формата

100 FORMAT (' STRING').

В некоторых случаях кроме ширины поля и инструкций, что делать с его содержимым, необходимо передать дополнительные инструкции устройству ввода-вывода. Для этой цели служат *управляющие дескрипторы*:

– дескрипторы BN и BZ используются для изменения способа обработки пробелов во входном файле. Вне зависимости от значения спецификации BLANK = в операторе OPEN, дескриптор BN

определяет обработку пробелов как пустых (отсутствующих) символов, BZ – как нулей только в данной спецификации формата;

– дескрипторы SS, SP и S; дескриптор SS подавляет печать знака «+» у положительного числа, SP – выводит «+» на печать, S – восстанавливает исходные условия вывода по умолчанию;

– дескриптор «/» прекращает обработку текущей записи и переходит к следующей записи при вводе или начинает новую запись (строку) – при выводе;

– дескриптор «:» прерывает выполнение спецификации формата, если в списке ввода-вывода не осталось необработанных элементов;

– дескрипторы табуляции (T*n*, TL*n*, TR*n*). Если текущая позиция записи (строки) – *k*, то дескриптор T*n* сменит текущую позицию на *n*, дескриптор TL*n* сместит текущую позицию влево на *n* символов, а TR*n* – тоже на *n* позиций, но вправо. Дескриптор *n*X выполняет ту же функцию, что и TR*n*.

Если файлы записываются и читаются на одном и том же компьютере, имеет смысл использовать бесформатный ввод-вывод. В этом случае величины записываются и читаются непосредственно во внутреннем представлении, без затрат на конверсию к внешнему представлению и, как следствие, без погрешностей округления. При последовательном бесформатном вводе-выводе в операторе OPEN спецификатору FORM= должно быть присвоено значение 'UNFORMATTED', а у операторов READ и WRITE должен отсутствовать спецификатор FMT=. Каждый из операторов READ или WRITE обрабатывает (передает) только одну запись. Обрабатываемый файл может быть только внешним.

ГЛАВА 2
ПРАКТИЧЕСКИЕ ЗАДАНИЯ

Практическая работа № 1.
Вычисление значения функции

Содержание лекции

Введение. Цели и задачи курса. Краткое знакомство с ОС Linux. Основные элементы языка FORTRAN, типы и длины величин, переменные и их описание. Арифметические операции. Встроенные функции. Ввод и вывод данных в свободном формате.

Рекомендуемые разделы учебников: глава 2 «Формулы» [4], глава 1 «Основы программирования на ФОРТРАНЕ» [5], электронные ресурсы [6, 7].

Программа 1. Вычисление значения функции

Задание

1. Вычислить значения функций:

а) $f(x,y)=\sqrt{\dfrac{7.5x+y}{3.7}}$,

б) $f(x)=sin(10x)/2$,

в) $f(x,y)=\sqrt[3]{x+y}$,

г) $f(x)=\dfrac{1}{2\sqrt{2}}(2-x)e^{-x/2}$,

д) $f(x)=-8.31\cdot 298.15\cdot \ln(x)$.

2. Найти значения постоянных *e* и *π* при помощи встроенных функций *dacos* (…) и *dexp* (…).

Указания

1. Значения переменных задайте самостоятельно.

2. Организуйте диалоговый ввод значений переменных *x* и *y* и вывод результата на консоль. Ввод переменных должен предваряться запросом пользователю на ввод.

3. Проведите контрольный расчет с помощью калькулятора. Сравните результаты.

4. Организуйте вывод результата в свободном формате.

5. *В рамках данной практической работы и в последующих необходимо использовать вещественные и комплексные переменные и константы двойной длины (двойной точности).*

6. *Современные компиляторы по умолчанию не различают прописные (заглавные) и строчные буквы (не различают верхний и нижний регистр).* Поэтому в дальнейшем можно использовать строчные буквы вместо прописных для написания программ на языке FORTRAN.

Входные параметры: значения *x* и *y* для выбранной функции.

Программа 2. Вычисление средней длины свободного пробега молекул газа

Задание

Вычислить среднюю длину свободного пробега молекул (или атомов) газа с известным эффективным диаметром при нормальных условиях. Рассмотреть три случая:

а) $d\,(N_2) = 3.20$ Å,

б) $d\,(Ar) = 3.64$ Å,

в) $d\,(H_2O) = 3.0$ Å.

Дополнительный материал

В рамках молекулярно-кинетической теории взаимодействие молекул рассматривается как упругое соударение шаров эффективного диаметра. Среднее расстояние, проходимое молекулой между двумя последовательными соударениями с другими молекулами (средняя длина свободного пробега), вычисляется по формуле:

$$\lambda = \frac{1}{\sqrt{2}\pi d^2 n}, \qquad (1)$$

где n – число молекул в единице объема (объемная плотность); d – эффективный диаметр молекулы, отвечающий минимальному расстоянию, на которое могут сблизиться центры масс

двух молекул. Число молекул в единице объема можно вычислить по известным давлению и температуре. Соответствующая формула выводится из уравнения состояния идеального газа

$$pV = \nu RT = \frac{N}{N_A} RT = N \frac{R}{N_A} T = NkT$$

и определения самого числа $n = N/V$, т. е.:

$$n = \frac{p}{kT}. \qquad (2)$$

Здесь N – число молекул газа, V – объем, N_A – число Авогадро, p – давление, T – температура, ν – количество вещества, k – постоянная Больцмана, R – газовая постоянная.

Указания

1. Повторите материал, связанный с явлениями переноса и длиной свободного пробега молекул из курса физической химии [8].
2. Организуйте диалоговый ввод и вывод, с необходимыми пользователю комментариями, включая и единицы измерения рабочих величин.
3. Расчет необходимо проводить в системе единиц СИ. Значения давления (в Па) и температуры (в К) при нормальных условиях можно определить внутри программы как константы.
4. Эффективный диаметр молекулы можно непосредственно ввести в единицах СИ или осуществить перевод из внесистемных единиц в программе.
5. Представьте число π как вещественную константу двойной длины *pi* = 2.d0*dacos (0.d0) (см. предыдущее задание).
6. Для расчета числа n запрограммируйте формулу (2). Постоянную Больцмана можно округлить до значения $1.38 \cdot 10^{-23}$ Дж/К.

Входные параметры: эффективный диаметр молекулы (атома) газа.

Практическая работа № 2.
Циклы, форматный ввод/вывод, оформление таблиц

Содержание лекции

Оператор цикла и оператор безусловного перехода. Чтение и запись информации в файл. Оформление выходных данных в удобном для работы виде: форматы представления переменных. Оператор FORMAT. Таблицы. Применение программы визуализации данных XMGrace. Блок-схемы программ.

Рекомендуемые разделы учебника: глава 2 «Формулы», глава 4 «Циклы» [4].

Программа 1. Построение таблицы

Задание

Построить таблицу зависимости массы кислоты от объема ее раствора при заданных массовой доле и плотности. Объем варьируется в интервале от 10 до 100 мл с шагом в 10 мл. Считать переменными: название вещества, массовую долю и плотность.

Дополнительный материал

Для вычисления массы вещества в растворе ($m_в$) при известных массовой доле вещества (W), плотности раствора ($\rho_{р\text{-}ра}$) и объеме раствора ($V_{р\text{-}ра}$) используется формула:

$$m_в = W \rho_{р\text{-}ра} V_{р\text{-}ра}. \quad (3)$$

Указания

1. Рекомендуется повторить способы задания концентрации раствора [9, 10].
2. Организуйте диалоговый ввод данных с необходимыми пользователю комментариями.
3. Сначала постройте таблицу для HCl с параметрами: $W = 12.51\,\%$, $\rho_{р\text{-}ра} = 1.060$ г/мл. Затем для H_2SO_4 с самостоятельно выбранными параметрами. Можно использовать справочник [10].
4. Организуйте вывод данных в файл с названием output. Файл должен содержать заголовок с контрольной выдачей входных данных, подзаголовки и саму таблицу. Например:

Table. Mass of acid (наименование кислоты) for W = (величина массовой доли) % and *p* = (величина плотности) g/ml.
Solution volume [ml] Mass of acid [g]

10.00 24.13
.....
100.00 125.11

Столбцы таблицы и их заголовки должны строго соответствовать друг другу (заголовки должны находиться точно над столбцами таблицы). Для этого рекомендуется осуществить первый запуск программы с пробным числом пробелов между наименованиями подзаголовков. Затем провести подсчет необходимых пробелов в получившемся выходном файле, переоформить оператор FORMAT, скомпилировать и выполнить (запустить) новую версию программы.

5. Придумайте числовой формат, который будет оптимальным для табличной записи объема раствора (10–100 мл) и массы кислоты (оцените необходимое число позиций целой и дробной частей с помощью калькулятора).

6. Вывод данных строки таблицы необходимо организовать в цикле.

Входные параметры: название вещества, массовая доля и плотность.

Программа 2. Функция распределения Максвелла

Задание

Построить функцию распределения Максвелла для молекул N_2 при температуре 300 K на интервале скоростей (10.0–1500.0) м/с.

Дополнительный материал

Функция распределения Максвелла для 1 моль газообразного вещества в приближении идеального газа имеет вид:

$$F(v) = A \cdot exp\left(-\frac{Mv^2}{2RT}\right) \cdot 4\pi v,\ A = \sqrt{\left(\frac{M}{2RT\pi}\right)^3} = \left(\frac{M}{2RT\pi}\right)^{3/2}. \qquad (4)$$

Здесь M – молярная масса молекулы (кг/моль), m – масса молекулы (кг), R – газовая постоянная, k – постоянная Больцмана, v – скорость молекул. Отношение M/R без привязки к 1 моль вещества можно заменить на m/k.

Указания

1. Рекомендуется повторить разделы «Статистические распределения. Распределение Максвелла» и «Основные понятия статистической термодинамики» в учебниках [8] и [11] соответственно.
2. Организуйте ввод данных из текстового файла input. Список входных параметров указан в разделе «Входные параметры».
3. Организуйте вывод данных в файл output. Файл должен содержать заголовок, контрольную выдачу входных параметров с указанием единиц измерения, подзаголовки и собственно таблицу с результатами вычислений.
4. Скопируйте полученный файл «output» в файл с расширением dat. В этом файле оставьте только таблицу с результатами вычислений.
5. График функции $F(v)$ можно построить с помощью программы *XMGrace* [12]. Для этого с помощью левой клавиши мыши необходимо выбрать в меню последовательность команд *Data* → *Import* → *ASCII* и выделить созданный файл с расширением *dat*.
6. Распечатать график можно, выбрав в меню последовательность команд *File* → *Print Setup* → *Device* – *JPG* (или *PNG*), *File name* – ввести имя файла и нажать кнопку *Accept*. Далее в меню нужно выбрать *File* → *Print*. В папке *Documents* или в домашней папке должен появиться сохраненный файл. Для сохранения в другом месте необходимо воспользоваться кнопкой *Browse*.
7. Внутри программы молярную массу необходимо перевести в единицы системы СИ.
8. Универсальную газовую постоянную можно взять равной 8.31441 Дж/(моль · К).

Входные параметры: температура в К, молярная масса молекулы в г/моль, нижняя и верхняя границы диапазона скоростей молекул в м/с, количество значений функции внутри диапазона скоростей.

Практическая работа № 3. Численное интегрирование

Содержание лекции

Интегрирование методами прямоугольников, трапеций и парабол. Сходимость интегралов и выбор критерия сходимости. Абсолютная и относительная ошибки интегрирования. Подпрограмма-процедура и подпрограмма-функция. Оператор условного перехода IF… END IF.

Рекомендуемые разделы учебника: глава 5 «Интегрирование» [4], глава LX «Численное интегрирование» [13].

Программа 1. Сравнение простейших способов интегрирования

Задание

На примере вычисления любого табличного интеграла сравните методы прямоугольников, трапеций и парабол. Результаты расчетов занесите в сравнительную таблицу (табл. 7).

Таблица 7

Сравнение эффективности работы различных методов интегрирования

Число точек интегрирования (npoint)	Значение интеграла, полученное по методу		
	прямоугольников	трапеций	парабол
Точное значение табличного интеграла:			

Желательно, чтобы число точек интегрирования для каждого метода совпадало.

Указания

1. Организуйте диалоговый ввод входных данных: выбор метода интегрирования, интегрируемой области (отрезка) и шага интегрирования (или числа узлов интегрирования).

2. Каждый из методов интегрирования оформите в виде подпрограммы, входными параметрами которой будут пределы интегрирования (*a, b*) и число точек интегрирования (*npoint*). На выходе подпрограмма должна передавать основной программе значение интеграла (*zint*).

3. Интегрируемую функцию оформите как подпрограмму-функцию.

4. Для выбора метода интегрирования в диалоговом режиме можно использовать блочный оператор условного перехода IF… END IF, который будет вызывать требуемую подпрограмму, проверяя значение введенной пользователем некоторой целой переменной. Например, методу прямоугольников можно сопоставить число «1», методу трапеций – «2», а методу парабол – «3».

5. После вывода полученного значения интеграла на консоль программа должна дать пользователю запрос на изменение числа точек интегрирования.

6. При проведении вычислений число точек интегрирования необходимо увеличивать до получения семизначных цифр.

7. Создайте в рабочей папке файл main.f, а для подпрограмм – файлы: Simple.f, Eyler.f, Simpson.f, fun.f. При компиляции исходного кода не забудьте указать все эти файлы.

Входные параметры: метод, пределы и число точек интегрирования.

Программа 2. Вычисление теплоемкости по формуле Дебая

Задание

Используя метод Симпсона, вычислить молярную теплоемкость C_v серебра при температуре $T = 173$ K. Характеристическую температуру серебра полагать равной $\Theta = 214$ K. Сколько точек интегрирования потребуется для достижения относительной ошибки $1 \cdot 10^{-5}$%?

Дополнительный материал

1. Теплоемкость – это количество теплоты, которое необходимо для нагревания единицы массы вещества на 1 K. В расчете на 1 моль вещества говорят о молярной теплоемкости. Так называемая истинная молярная теплоемкость представляется в виде:

$$C = \frac{\delta Q}{dT} \text{ [Дж/(моль·К)]}, \quad (5)$$

где δQ – бесконечно малое количество теплоты, dT – бесконечно малое приращение температуры. Истинная молярная теплоемкость при постоянных объеме и давлении, в свою очередь, это:

$$C_V = \left(\frac{\partial U}{\partial T}\right)_V, \; C_P = \left(\frac{\partial H}{\partial T}\right)_P, \quad (6)$$

где U и H – внутренняя энергия и энтальпия соответственно. Известна также связь между C_V и C_P:

1) для твердых и жидких тел: $(C_P - C_V) \to 0$;

2) для идеального газа: $(C_P - C_V) = R$.

Теплоемкость твердых веществ может быть оценена по формуле Дебая [14, 15]:

$$C_V = \frac{9R}{Z^3}\int_0^Z x^4 \frac{e^x}{(e^x - 1)^2} dx, \quad Z = \frac{\theta}{T}, \quad (7)$$

где $R = 8.31441$ Дж/(моль К); θ – характеристическая температура (иногда называемая «температурой Дебая»), зависящая от природы вещества. Значения подынтегрального выражения (7) традиционно приводятся в справочниках, но иногда требуется расчет самого интеграла.

2. Относительная ошибка:

$$\delta = \left|(a_{exact} - a_{approximate}) \cdot 100 / a_{approximate}\right| \text{ [\%]}, \quad (8)$$

где $(a_{exact} - a_{approximate}) = \varepsilon$ – абсолютная ошибка, являющаяся разницей между точным значением некоторой величины a и ее приближенным значением. В том случае, когда абсолютная ошибка мала, точное значение в знаменателе формулы (8) можно заменить приближенным:

$$\delta = \left| \left(a_{exact} - a_{approximate} \right) \cdot 100 / a_{approximate} \right|. \quad (9)$$

Далее будем полагать, что приближенное значение – это значение интеграла на настоящем шаге цикла (*zint2*), а точное значение – значение интеграла на предыдущем шаге цикла (*zint1*), тогда относительную ошибку можно оценить по формуле:

$$\delta = \left| \left(zint1 - zint2 \right) \cdot 100 / zint2 \right|. \quad (10)$$

Указания

1. Рекомендуется повторить раздел физической химии «Термодинамика», в частности можно использовать главу IIБ второй части учебника [16].

2. Как и в предыдущем задании практической работы, оформите расчет интеграла методом парабол (методом Симпсона) в виде подпрограммы и поместите ее в отдельный файл (Simpson.f). Интегрируемую функцию удобно оформить как подпрограмму-функцию (файл fun.f).

3. В том случае, если требуемая точность расчета интеграла не была достигнута при заданном пользователем числе узлов интегрирования (*npoint*), значение *npoint* необходимо увеличить и повторить расчет интеграла. Для этого удобно организовать цикл с помощью блочного оператора ветвления IF/ENDIF и оператора безусловного перехода GOTO. Начальное значение переменной *zint1* можно положить равным единице.

4. После расчета интеграла и относительной ошибки нужно организовать контрольную выдачу значений интеграла на предыдущем и настоящем шагах цикла вместе с относительной ошибкой и числом узлов интегрирования.

5. Чтобы избежать попадания в особую точку интегрируемой функции, в качестве левой границы области интегрирования необходимо выбрать малое положительное число, например $a = 0.00001$.

6. Ввод и вывод необходимо оформить в диалоговой форме.

*Входные параметры: х*арактеристическая температура, температура и количество точек интегрирования. *npoint* = 10, 100, 200, 400, …, в зависимости от значения требуемой относительной ошибки вычисления интеграла.

Практическая работа № 4.
Решение нелинейных уравнений

Содержание лекции

Вычисление значений многочленов. Численные методы решения нелинейных уравнений. Метод Ньютона (метод касательных), метод деления отрезка пополам. Недостатки этих методов. Использование внешних (библиотечных) подпрограмм (zbrak и rtbis в программе деления отрезка пополам). Оператор описания EXTERNAL. Индексируемые переменные. Массивы. Сортировка исходных файлов по папкам.

Рекомендуемые разделы учебников: глава LVIII «Нелинейные уравнения и системы» [13], глава 6 «Уравнения» [4], глава 9 «Root finding and nonlinear sets of equations» [17], § 3.5 «Вычисление ряда» [5].

Программа 1. Метод Ньютона

Задание

Найдите решения уравнения $10x^5 - e^{x^2} = 0$ методом Ньютона (методом касательных).

Указания

1. Оформите вычисление функции $y = 10x^5 - e^{x^2}$ и ее производной как подпрограммы-функции. Текст этих подпрограмм можно сохранить как в отдельном файле, так и в файле с текстом основной программы.

2. Используйте в качестве критерия сходимости абсолютное значение разности между значениями *x* на предыдущем и настоящем шаге. Возьмите в качестве порога сходимости число $\varepsilon = 10^{-8}$.

3. В качестве начального значения переменной можно взять $x = 1$ и $x = 5$. В качестве альтернативы можно выбрать любое значение *x*, близкое к точке смены знака функции. При его выборе необходимо учесть, что начальное значение *x* может соответствовать точке с близким или равным нулю значением производной, что приведет к делению на нуль в рекуррентной формуле метода Ньютона. Как избежать данной проблемы?

4. Чтобы избежать зацикливания программы можно вести учет числа итераций программы с принудительным ее остановом по достижению предельного числа итераций.

5. Приведите примеры ситуаций, в которых разработанная программа будет работать некорректно.

Входные параметры: начальное значение переменной *x*.

Программа 2. Метод деления отрезка пополам

Задания

1. Найдите решения уравнения $10x^5 - e^{x^2} = 0$ методом деления отрезка пополам (*бисекции*).

2. Вычислите объем 1 моль насыщенного водяного пара при 485 К и давлении 2.052 МПа (20.25 атм) в приближении Ван-дер-Ваальса (формулу см. ниже). Параметры уравнения Ван-дер-Ваальса для воды: $a = 5.464$ л^2·атм/моль^2, $b = 0.03$ л/моль (используется задача № 6 главы IX задачника [18]). Значение универсальной газовой постоянной $R = 0.082$ л·атм/(К·моль).

Дополнительный материал:

1. Уравнение Ван-дер-Ваальса для *n* моль газа можно записать в виде:

$$\left(p + \frac{n^2 a}{V^2}\right)(V - nb) = nRT, \tag{11}$$

где V – объем; p – давление; T – температура; n – количество вещества; R – газовая постоянная; a – постоянная, учитывающая взаимное притяжение молекул; b – постоянная, учитывающая собственный объем молекул.

2. Умножим обе части равенства (11) на V^2, после чего перенесем правую часть уравнения влево и приведем подобные слагаемые. В результате получим уравнение:

$$f(V) = pV^3 - (pb + RT)V^2 + aV - ab = 0. \tag{12}$$

3. При вычислении полинома $f(V)$ (12) необходимо осуществить возведение в степень, что может привести к потере точности. Чтобы сохранить точность расчета и сократить объем вычислений, необходимо воспользоваться правилом Горнера (так называемый метод «деления уголком»). Подробнее см. в § *3.5 «Вычисление*

ряда» [5]. Для этого перепишем слагаемые в порядке возрастания степеней *V*:

$$f(V) = -ab + aV - (pb + RT)V^2 + pV^3. \quad (13)$$

И наконец, заменим возведение в степень умножением, тогда запись по правилу Горнера:

$$f(V) = -ab + V(a - V(pb + RT - pV)). \quad (14)$$

Указания

1. Рекомендуется повторить раздел физической химии, посвященный термодинамике идеальных и реальных газов: глава 1, § 1 «Основные понятия термодинамики. Уравнения состояния» [11], глава IX «Газы» [18].

2. Для работы используйте алгоритмы ZBRAK и RTBIS из [17] § *9.1 bracketing and bisection.* Их блок-схемы приведены в Приложении 1.

3. Функцию (14) оформите в виде подпрограммы-функции с именем FUNC.

4. В головной программе и подпрограммах укажите FUNC как внешнюю функцию с помощью оператора EXTERNAL.

5. Одномерные массивы должны иметь одинаковые размеры. Можно положить размер одномерного массива равным 10.

6. Организуйте программу таким образом, чтобы входные параметры вводились из файла, а выходные – выводились на терминал (консоль) в произвольном формате.

7. Все файлы с исходным кодом поместите в папку с названием src (source – исходный код). Создайте папку bin, в которую будет помещен исполняемый файл.

8. В рабочем каталоге с папками src и bin должны находиться входной файл input и исполняемый командный файл оболочки, отвечающий за сборку программы, – xx. Файл xx должен иметь следующее содержание:

gfortran -o./bin/bisection.x./src/.f*
exit

Данная команда при ее исполнении компилирует все файлы с расширением .f, находящиеся в папке src, и помещает исполняемый файл bisection.x в папку bin.

9. Файл xx нужно сделать исполняемым с помощью команды chmod u + x xx.

10. Для сборки программы потребуется лишь запустить xx и проверить наличие бинарного файла bisection.x в соответствующей папке или прочесть сообщения компилятора в случае ошибки.

11. Для запуска программы **из рабочего каталога** нужно использовать команду ./bin/bisection.x.

12. Подобное структурирование исходных и исполняемых файлов обретает дополнительный смысл особенно в тех случаях, когда для работы требуется значительное количество подпрограмм.

13. Попробуйте изменить входные параметры и проследите, как изменится при этом ответ. В каком случае он будет точнее? В каком случае будут найдены не все три корня?

14. После проведения расчета определите, какой из трех корней будет иметь физический смысл искомой величины.

Входные параметры: левая и правая границы интервала поиска значения искомой переменной; количество подынтервалов, на которые первоначально необходимо разбить исходный интервал; максимальное количество корней уравнения; точность расчета (необходима для завершения итерирования) – все вводятся из файла «input».

Практическая работа № 5.
Решение дифференциальных уравнений первого порядка

Содержание лекции

Методы численного решения дифференциальных уравнений: Эйлера, Рунге – Кутты. Задача Коши. Начальные и краевые условия. Решение системы дифференциальных уравнений первого порядка с начальными условиями методом Эйлера.

Рекомендуемые разделы учебников: главы XXI и XXIII «Дифференциальные уравнения первого порядка» и «Системы дифференциальных уравнений» [19], глава LXII «Обыкновенные дифференциальные уравнения. Задача Коши» [13], глава 9 «Дифференциальные уравнения» [4], глава 16 «Integration of Ordinary Differential Equations» [17].

Программа 1. Решение дифференциального уравнения методом Эйлера

Задания

1. С помощью метода Эйлера найдите частное решение дифференциального уравнения $y'(x^2-4)=2xy$ с начальным условием $y(0)=1$ на интервале $0 \le x \le 1$. При решении найдите величину шага интегрирования, позволяющего воспроизвести значение функции в точке *y*(1) с тремя точными значащими цифрами. После этого постройте график функции *y*(*x*) на интервале $0 \le x \le 1$.

2. Охлаждение некоторого физического тела, нагретого до 100 °C в помещении с комнатной температурой (20 °C), происходит по закону $dT(t)/dt = -k(T(t)-20)$, где $k = 0.0693$, T – температура, t – время. До какой температуры охладится тело спустя 10 мин; 40 мин? Точность решения задайте самостоятельно.

Указания

1. Организуйте ввод данных в диалоговом режиме.

2. Значения *x* и *y* сохраняйте в одномерных массивах.

3. Зарезервируйте возможность увеличивать число точек дифференцирования для достижения требуемой точности вычислений. Для этого используйте операторы IF/ENDIF и GOTO аналогично тому, как это было сделано в Практической работе № 3 (программа 2). Не забудьте о промежуточных контрольных выдачах.

4. Введите критерий сходимости как абсолютную величину разности значения функции *y* в некоторой точке при заданном числе точек дифференцирования и значения функции *y* для числа точек дифференцирования, взятого на предыдущей итерации. Стартовое значение *y* положите равным единице. Положите пороговое значение критерия сходимости *epsilon* равным 5.d-4.

5. По достижении заданной точности сохраните значения функции в файле с расширением dat – для построения графика в *XMGrace* (см.: Практическая работа № 2, программа 2).

6. Вычисление производной функции *y* можно реализовать как отдельную подпрограмму-функцию. Текст подпрограммы можно сохранить в том же файле, что и основную программу.

7. Формат выдачи данных в файл выберите самостоятельно.

Входные параметры: начальные значения *x* и *y;* конечное значение *x*, для которого нужно отыскать значение функции; число точек дифференцирования (*npoint* = 50, 100, 200, 400…).

Программа 2. Решение системы дифференциальных уравнений методом Эйлера

Задание

Решите систему уравнений для последовательных химических реакций $k = 0.0693$. Значения констант скоростей реакций положите равными $k_1 = 11 \cdot 10^{-3}$ с$^{-1}$, $k_2 = 1 \cdot 10^{-3}$ с$^{-1}$.

а) Найдите концентрации веществ *A, B, E* спустя 10 с после начала химической реакции, если в начальный момент времени *c(A)* = 1 моль/л.

б) Постройте кинетические кривые для процесса, протекающего в течение одного часа.

Дополнительный материал

Кинетика сложных реакций описывается на основе кинетики элементарных реакций. Элементарная реакция – это единичный акт образования или разрыва химической связи, протекающий через образование переходного комплекса.

Каждая элементарная реакция независимо от других подчиняется закону действующих масс, т. е. для процесса $n_1 A + n_2 B \xrightarrow{k} C$ верно равенство $\frac{d[A]}{dt} = \frac{d[B]}{dt} = -k[A]^{n_1}[B]^{n_2} = \frac{-d[C]}{dt}$. Применительно к имеющейся задаче последовательных реакций получаем систему кинетических уравнений:

$$\begin{cases} \dfrac{d[A]}{dt} = -k_1[A], \\ \dfrac{d[E]}{dt} = k_1[A] - k_2[E], \\ \dfrac{d[B]}{dt} = k_2[E]. \end{cases} \qquad (15)$$

Указания

1. Рекомендуется повторить разделы: «Кинетика химических реакций» (часть 5 [16]) и «Химическая кинетика» (глава 5 [11]).

2. Реализуйте расчет производных посредством подпрограммы-функции с сохранением концентраций в двумерном массиве *y*: $[A] = y(i, 1)$, $[E] = y(i, 2)$, $[B] = y(i, 3)$, где i – номер точки дифференцирования. Значения констант скорости необходимо задать в подпрограмме.

3. Организуйте диалоговый ввод для числа точек дифференцирования (аналогично предыдущей программе) и номера уравнения, по которому будет проводиться контроль сходимости (параметр *itest*). Для ввода прочих данных используйте файл input. Концентрации веществ в начальный момент времени можно присвоить элементам одномерного массива.

4. Предусмотрите возможность изменения числа точек дифференцирования для достижения требуемой точности вычислений без выхода из программы. Определение точности расчета следует сделать по каждому из уравнений системы (15) (сначала параметр *itest* взять равным 1, потом – 2, потом – 3). По результатам расчета необходимо выбрать максимальное из использованных число точек и сохранить для него ответ для каждого уравнения.

5. В качестве критерия сходимости решения возьмите абсолютное значение разности значений функции *y(t)* в выбранной точке, найденных на текущей и предыдущей итерациях. На первой итерации стартовое значение *y(t)* положите равным единице. Значение критерия сходимости (*epsilon*) положите равным 5.d-4.

6. Определите концентрации всех веществ через 10 с и через 1 ч после начала реакции. Выбор числа точек интегрирования в обоих случаях должен быть проведен независимо.

7. Организуйте вывод времени и концентраций веществ (кинетических кривых) в файл в виде таблицы. Используйте форматный вывод. Вывод данных можно реализовать с помощью *вложенного* цикла, например:

```
do i = 1, npoints
write (5,100) x (i), (y (i, j), j = 1, numeq)
end do
```

где *npoints* – количество выводимых в файл строк (точек интегрирования), *numeq* – число уравнений в системе; *x* и *y* – массивы, содержащие значения независимой переменной и концентраций

веществ. Выводимая в файл строка будет содержать элемент *x(i)* и элементы с *y*(*i, 1*) по *y*(*i, numeq*).

8. Используйте полученный выходной файл для построения кинетических кривых с помощью *XMGrace*. Введите обозначения для осей координат, добавьте легенду. Для этого необходимо самостоятельно ознакомиться с описанием программы *XMGrace*.

9. Определите, в какой момент времени: а) концентрации веществ *E* и *B* равны; б) концентрации веществ *A* и *B* равны; в) максимальна концентрация промежуточного продукта. Достигли ли концентрации веществ за один час времени своих равновесных значений?

10. В данной задаче химическая реакция лимитируется второй стадией. Проверьте, как изменится вид кинетических кривых, если задать константы скоростей $k_1 = 1 \cdot 10^{-3}$ с$^{-1}$ и $k_2 = 11 \cdot 10^{-3}$ с$^{-1}$.

Входные параметры: начальное и конечное значения интервала дифференцирования, число уравнений в системе и начальные значения концентраций – в файле input; число точек дифференцирования (*npoint* = 10, 20, 50, 100, 200, 300...), параметр *itest* – в диалоговом режиме.

Практическая работа № 6. Метод Монте-Карло

Содержание лекции

Метод Монте-Карло. Его применение в физике и химии. Генераторы случайных и псевдослучайных чисел. Встроенные функции random_seed и random_number языка FORTRAN.

Рекомендуемые разделы учебников: § *4.4 «Интегрирование методом Монте-Карло»,* § *7.2 «Интегрирование кинетического уравнения первого порядка методом Монте-Карло» [4],* § *5.3 «Методы численного эксперимента» [20], глава 7 «Random numbers» [17].*

Программа. Моделирование последовательных реакций первого порядка методом Монте-Карло

Задание

Провести моделирование системы последовательных реакций: $A \xrightarrow{k_1} E \xrightarrow{k_2} B$. Значения констант скоростей реакций возьмите

равными: $k_1 = 11 \cdot 10^{-3}$ с$^{-1}$, $k_2 = 1 \cdot 10^{-3}$ с$^{-1}$. Требуется получить кинетические кривые данного процесса для временного интервала в 1 ч. Сравнить результат с результатами расчета методом Эйлера, которые были получены в предыдущей работе. Представить на одном графике кинетические кривые, построенные с помощью этих двух методов.

Указания

1. Исходя из имеющихся величин констант скорости k_1 и k_2, нужно ответить на вопросы:

а) Каково общее количество частиц? За их число можно взять общий знаменатель у обеих констант, если перевести значения констант в дроби. В начале процесса это число будет совпадать с исходным количеством частиц только типа *A* (входной параметр *n*0).

б) Сколько частиц типа *A* от этого общего количества прореагировало (а соответственно, образовалось частиц типа *E*) за 1 с? И сколько частиц типа *E* также от общего количества прореагировало за одну секунду с образованием частиц типа *B*? Это будут входные параметры (назовем их, например, nk_1 и nk_2) – в качестве этих чисел можно взять числители указанных выше дробей.

в) Сколько времени длится моделируемый процесс (входной параметр *ntime*)?

г) Какова исходная молярная концентрация вещества *A* (входной параметр *C*0)?

2. Не забудьте создать двумерный массив *ny*, который будет содержать информацию о существующих частицах. При этом первый индекс элемента массива будет отвечать за номер частицы, а второй индекс – за тип (1 – частицы *A*, 2 – *E*, 3 – *B*). Пусть содержимое ячейки массива, равное 1, будет говорить о существовании частицы данного типа, а содержимое, равное 0, – о том, что такой частицы еще нет или уже нет. Также не забудьте задать исходные значения массива в самом начале программы.

3. Размер массива *ny* должен соответствовать параметру *n*0. В противном случае результат расчета будет неверным. Для этого, например, можно все вычисления провести внутри подпрограммы, в которой массив *ny* сделать регулируемого размера, а *n*0 станет формальным параметром подпрограммы. Это также позволит задать *n*0 как входной параметр, что достаточно удобно.

4. Для ввода и вывода данных используйте файлы input и output.dat.

5. Увеличьте значения входных параметров nk_1, nk_2, $n0$ в 2, 5, 10, 25 раз и посмотрите на результат. Изменилась ли точность расчета? Уменьшились ли амплитуды осцилляций? Как соотносятся результаты, полученные методом Монте-Карло и методом Эйлера? Каковы причины этих результатов?

Входные параметры: время протекания процесса (с), начальная концентрация вещества *A* (моль/л), nk_1, nk_2 и общее количество частиц (вводятся из файла input).

Практическая работа № 7. Решение системы линейных уравнений

Содержание лекции

Матрицы. Методы решения систем линейных уравнений: Гаусса, Гаусса – Жордана, LU-разложение с обратным ходом.

Рекомендуемые разделы учебников: глава IV «Матрицы. Определители. Линейные системы» [21], глава 6 «Линейные системы» [4], § 2.3 «LU Decomposition and Its Applications» [8]. Используйте также электронные ресурсы [22]. Краткое описание метода Гаусса представлено в Приложении 2.

Программа. LU-разложение с прямым и обратным ходом

Задания

1. Решите систему линейных уравнений

$$\begin{cases} x_1 - x_2 + 2x_3 = 2, \\ x_1 + 2x_2 + x_3 = -2, \\ 2x_1 - x_2 - x_3 = -2 \end{cases}$$

с помощью LU-разложения с прямым и обратным ходом. Для решения используйте стандартные процедуры *DGETRF* и *DGETRS* пакета *LAPACK* и используемые ими подпрограммы.

2. Даны четыре емкости с растворами кислоты различной концентрации. Если смешать растворы в определенном соотношении, то получится раствор кислоты известной объемной концентрации (табл. 8).

Таблица 8

Объемное соотношение и объемная концентрация кислоты

Объемное соотношение	Объемная концентрация, %
1 : 1 : 1 : 1	25
4 : 3 : 2 : 1	20
4 : 1 : 1 : 4	25
4 : 1 : 4 : 1	22

Определите, какова концентрация кислоты в каждой из емкостей (используется задача № 99 главы 6 учебника [4]).

Дополнительный материал

1. Каждая из рассматриваемых систем линейных уравнений может быть представлена в виде матричного уравнения: $\hat{A}\vec{x} = \vec{B}$, где $\hat{A}$ – матрица коэффициентов, $\vec{x}$ – искомый вектор-столбец переменных, $\vec{B}$ – вектор свободных членов.

2. Программу для решения системы линейных уравнений всегда можно написать самостоятельно, но также необходимо уметь пользоваться и стандартными библиотеками. Наиболее известные из них:

BLAS – Basic Linear Algebra Subprograms – библиотека базовых подпрограмм линейной алгебры (умножение, сложение матриц и т. д.), входит в состав пакетов процедур *LAPACK*, *ARPACK*;

LAPACK (Linear Algebra PACKage) – пакет процедур линейной алгебры, разработанных на базе проекта *LINPACK*.

3. Объемная концентрация вещества в растворе (в %) определяется по известной формуле: $\varphi = V(в-ва)/V(р-ра)\cdot 100\%$. Поэтому, прежде чем записывать значения элементов вектора $\vec{B}$, нужно сначала отыскать их значения по соответствующей формуле. Вычисления несложные, программировать их необязательно.

Указания

1. Исходные тексты и алгоритмы пакетов линейной алгебры *BLAS* и *LAPACK* можно найти на официальном портале Netlib [22].

Проведите поиск процедур DGETRF и DGETRS. В конце файла с исходным кодом найдите список используемых процедур, выпишите названия необходимых подпрограмм и также найдите их исходные тексты. Все исходные тексты положите в папки с соответствующими названиями. Описание двух основных процедур можно найти либо в скачанных файлах, либо в Приложении 3.

2. Проведите структурирование исходных файлов по папкам: файл с головной программой *main.f* поместите в папку src; процедуры DGETRF и DGETRS вместе с соответствующими зависимыми подпрограммами поместите в папку lapack; все процедуры элементарных матричных преобразований поместите в папку blas.

3. Для исполняемого файла создайте папку bin. В рабочем каталоге обеспечьте наличие файла xx для компиляции программы. О создании этого файла xx и его работе уже говорилось в Практической работе № 4 (программа 2).

4. Не забудьте, что для запуска программы **из рабочего каталога** нужно использовать команду *./bin/название_исполняемого_файла*

5. Ввод данных организуйте из файла input. Только размерность матрицы $\hat{A}$ вводите через консоль.

6. Процедуры *LAPACK* используют массивы регулируемого и предполагаемого размера. Однако перед работой этих процедур в головной программе уже приходится работать с данными массивами, размеры которых заданы постоянными. В результате может возникнуть несоответствие между размерами массивов, поскольку точный их размер определяется только в процессе работы программы. Чтобы процедуры *LAPACK* корректно работали, можно, например, запустить их из **отдельной подпрограммы**, в которой будут проводиться все операции с массивами. Так можно будет задать для них регулируемый размер, который будет полностью соответствовать размеру из процедур *LAPACK,* а информация о длине массива будет передаваться из головной программы.

7. Решение второго задания необходимо начать с составления системы линейных уравнений. Систему необходимо переписать в рабочую тетрадь.

8. Выберите величины, которые будут приняты за переменные x_i, составляющие вектор $\vec{x}$.

9. После решения задачи укажите значения искомых переменных и концентрацию кислоты в каждом из сосудов.

Входные параметры: элементы матрицы $\hat{A}$, пустая строка, элементы вектора $\vec{B}$ – вводятся из файла input. Размерность матрицы $\hat{A}$ – в диалоговом режиме.

Практическая работа № 8. Сплайн-интерполяция

Содержание лекции

Методы интерполяции. Интерполяционный полином. Сплайны: виды сплайнов, граничные условия, интерполяционный сплайн, сглаживающий сплайн.

Рекомендуемые разделы учебников: глава LIX «Вычисление значений функций», § 1 «Интерполяция многочленами», § 2 «Интерполяция кусочно-полиномиальными функциями», глава LXV «Сплайны», § 1 «Сплайн-функции» [13], глава 10 «Интерполяция» [4], глава 3 «Interpolation and extrapolation» [17].

Программа. Кубическая сплайн-интерполяция

Задание

Дана зависимость *y* от *x* для 21 значения *x*, полученная в результате расчета или эксперимента. Для данной зависимости постройте интерполированную кривую из 1000 точек, используя кубический сплайн. Исходная кривая выдается преподавателем в виде файла input.

Дополнительный материал

1. Программа для проведения сплайн-интерполяции состоит из двух последовательно используемых процедур: SPLINE и SPLINT. Первый вычисляет необходимые коэффициенты сплайн-функции на каждом отрезке табулированной функции. Второй вычисляет значения сплайн-функции в заданной точке *x*, лежащей внутри табулированного интервала, т. е. проводит собственно интерполяцию. Оба алгоритма можно написать самостоя-

тельно или взять из стандартных библиотек. В частности, можно использовать алгоритмы из § 3.3 «Cubic spline interpolation» [17]. Описание алгоритмов можно найти в Приложении 4.

2. Многие программы, подобные *XMGrace*, также способны проводить сплайн-интерполяцию. В *XMGrace* для этого нужно только выполнить: *Data* ⟶ *Transformations* ⟶ *Interpolation/Splines*. Однако в отсутствие исходного текста программы сложно оценить ее возможности.

Указания

1. Проведите структурирование исходных файлов по папкам: файл с головной программой *main.f* напишите самостоятельно и поместите в папку src; туда же положите процедуры SPLINE и SPLINT; для бинарного файла создайте папку bin. В рабочем каталоге обеспечьте нахождение файла xx для компиляции программы. О создании этого файла и его работе уже говорилось в Практической работе № 4 (программа 2).

2. Запуск программы **из рабочего каталога** осуществляется командой *./bin/название_исполняемого_файла*

3. Организуйте ввод данных из файла input, взятого у преподавателя (или используйте тот, что приведен ниже). Для ввода числа точек интерполяции используйте диалоговый режим.

4. Ознакомьтесь с описанием подпрограмм SPLINE и SPLINT (Приложение 4). Подпрограмма SPLINE при проведении интерполяции используется однократно, а SPLINT – в каждой интерполируемой точке. Для этого последнюю подпрограмму необходимо вызывать в цикле. При построении сплайна используйте естественные граничные условия.

5. При считывании из файла неизвестного числа строк удобно указать в операторе чтения файла спецификацию END = LABEL, с помощью которой по достижении конца файла будет совершен переход к оператору с меткой LABEL.

6. Сохраните интерполированную функцию в файле output.dat. Воспользовавшись *XMGrace* (см.: Практическая работа № 2, программа 2), сравните полученную кривую с исходной. Сохраните графики.

7. Сравните полученную кривую с интерполяционной кривой, полученной *XMGrace*.

Входные параметры
Файл *input*:
0.35322839 0.7285977E + 01
0.35589604 0.1306049E + 02
0.35749190 0.1278284E + 03
0.35798356 0.5870883E + 03
0.35817152 0.1218370E + 04
0.35827998 0.1836793E + 04
0.35835964 0.2311021E + 04
0.35842687 0.2614085E + 04
0.35848846 0.2770588E + 04
0.35854790 0.2815958E + 04
0.35860724 0.2781040E + 04
0.35866794 0.2688046E + 04
0.35873125 0.2550838E + 04
0.35879847 0.2376725E + 04
0.35887122 0.2167296E + 04
0.35895192 0.1918569E + 04
0.35904475 0.1621179E + 04
0.35915814 0.1263081E + 04
0.35931292 0.8410742E + 03
0.35957694 0.4102382E + 03
0.36027068 0.8038838E + 02
0.36242759 0.1003525E + 02
Количество точек интерполирования (интерактивно).

Практическая работа № 9. Метод наименьших квадратов

Содержание лекци

Метод наименьших квадратов (МНК), область его применения. Функция χ^2 и ее минимизация. Линейная регрессия; аппроксимация нелинейной функцией известного вида. Метод Левенберга – Марквардта.

Рекомендуемые разделы учебников: § *7.7 «Линейная регрессия»*, § *8.3 «Линейная регрессия общего вида»*, § *11.2 «Нелинейная регрессия» [4], глава 15 «Modeling of data» [17].*

Оригинальные статьи: [23, 24].

Программа 1. Аппроксимация линейной функцией

Задание

Разложение ацетондикарбоновой кислоты в водном растворе – реакция первого порядка. Измерены константы скорости этой реакции при разных температурах (табл. 9).

Таблица 9

Параметры реакции разложения ацетондикарбоновой кислоты

t, °C	0	20	40	60
k·10^5, c^{-1}	2.46	47.5	576	5480

Рассчитайте энергию активации и предэкспоненциальный множитель для уравнения Аррениуса с помощью аппроксимации линейной функцией методом МНК (используются данные из задач № 20–26 главы 5 «Химическая кинетика» учебника [11]).

Дополнительный материал

1. Скорость большинства реакций увеличивается с ростом температуры. В качестве одного из соотношений, которое может количественно описать температурные эффекты в химической кинетике, используют уравнение Аррениуса

$$k(T) = Ce^{-E_A/(RT)}, \tag{16}$$

где R – универсальная газовая постоянная, E_A – энергия активации (характеризует энергетический барьер на пути реакции), не зависит от температуры, C – предэкспоненциальный множитель, который также не зависит от температуры, а определяется только видом реакции; это доля молекул, энергия которых превышает E_A при температуре T.

Определить оба параметра можно, представив уравнение Аррениуса в логарифмической форме:

$$\ln(k) = \ln(C) - \frac{E_A}{RT}\cdot \tag{17}$$

Линейная аппроксимация вида

$$y = a + bx \qquad (18)$$

даст параметры a и b. Тогда соотношение величин между формулами (17) и (18) будет следующее:

$$y = \ln(k),\ x = 1/T,\ a = \ln(C),\ b = \frac{-E_A}{R}. \qquad (19)$$

2. *XMGrace* умеет проводить линейную регрессию. Для этого нужно выполнить команды *Data → Transformations →Regression*. В этом разделе *XMGrace* есть возможность проводить и нелинейное МНК некоторых стандартных видов, например полиномиальное или экспоненциальное. Ознакомьтесь с вариантами. Однако если в списке нет подходящей функции, то нужно открыть другой раздел меню: *Data →Transformations →Nin-linear curve fitting*. Об этом подробнее в следующем задании.

Указания

1. Оформите программу линейного МНК в виде подпрограммы. Для написания подпрограммы используйте математический аппарат, приведенный на лекции, или же воспользуйтесь § *7.7 «Линейная регрессия»* из учебника [4].

2. При создании подпрограммы потребуется использовать встроенную функцию FLOAT (***name***), которая преобразует целую величину стандартной длины ***name*** к вещественному типу, во избежание потери точности в процессе вычислений.

3. Число точек исходной кривой нужно определить с помощью программы – для этого воспользуйтесь оператором READ со спецификацией END (см.: Практическая работа № 8).

4. Проводить структурирование исходных файлов по папкам необязательно.

5. Для ввода исходных данных используйте файл input.

6. Программа должна преобразовать данные из файла input к расчетным единицам: температуру перевести в Кельвины, а константу скорости – в $см^{-1}$.

7. Организуйте вывод в соответствии с форматом следующего вида:

```
100 format (1x,'Linear list-squares fitting y = a + bx',/
. 1x,'a  = ', e14.7,1x,'b  = ', e14.7,/
. 1x,'Chi2  = ', e14.7)
```

101 format (1x,'C = ', e14.5,1x,'1/s',1x,'Ea = ', f10.5,1x,'kJ/mol')

8. Сравните результат, полученный вами, с результатом, полученным в *XMGrace*.

Входные параметры: зависимость константы скорости реакции от температуры (файл input).

Программа 2. Аппроксимация функцией Лоренца

Задание

Дана зависимость *y* от *x* для 21 значения *x*, полученная в результате расчета (эксперимента). Известно, что связь между переменными *x* и *y* описывается функцией Лоренца. С помощью метода Левенберга – Марквардта найдите параметры функции: ее ширину и положение пика. Для выполнения задания используйте редактор *XMGrace*. Исходная кривая выдается преподавателем в виде файла «input» (эта кривая может совпадать с кривой из Практической работы № 8).

Дополнительный материал

1. В общем виде функция Лоренца имеет вид

$$y(x) = \frac{a_0}{(x - a_1)^2 + \frac{a_2^2}{4}}, \qquad (20)$$

где a_1 – соответствует положению пика, a_2 – его ширине. Отметим, что в различных математических пакетах могут использоваться модифицированные формы выражения (20), например с дополнительным слагаемым, отвечающим за асимметрию графика.

2. Данное задание не требует программирования на FORTRAN, как это было во всех предыдущих заданиях. Однако самостоятельно вы можете попробовать использовать оригинальный открытый алгоритм из библиотеки MINPACK [25] или же алгоритм MRQMIN из § 15.5 «Nonlinear Models» [17].

3. Редактор *XMGrace* имеет встроенную процедуру нелинейного МНК – алгоритм Левенберга – Марквардта, основанный на подпрограмме LMDIF из упомянутого в предыдущем пункте пакета MINPACK. Ваша задача – задать функцию (20) *XMGrace*.

4. Нелинейная аппроксимация требует задания вида функции и пробных значений искомых параметров.

Указания

1. Используйте файл input из предыдущей практической работы. Откройте его в редакторе (*Data → Import → ASCII →* открываемый файл).

2. Выберете раздел нелинейной аппроксимации МНК *(Data → Transformations → Non-linear curve fitting).*

3. Задайте параметры аппроксимации. Для этого в левом окне *Set* выберите рабочую функцию *G0.S0.* Задайте ее форму: *y = a*0/((*x-a*1)^2 + *a*2^2/4)

Далее: Parameters – 3, *Tolerance* – 0.00001, *Iterations* – 10, *a*0 = 1.

4. Значения *a1 и a2* задайте самостоятельно. Для этого воспользуйтесь возможностями редактора *XMGrace* – он показывает в верхнем левом углу экрана координаты точки при наведении стрелки мыши.

5. Нажмите клавишу *Apply*. Красная кривая на графике – это результат аппроксимации. 10 итераций может оказаться недостаточно. Нажимайте клавишу *Apply* до тех пор, пока красная кривая не станет похожа на оригинал, а параметры *a*0, *a1 и a*2 не перестанут меняться.

6. Результат запишите в тетрадь:

*a*0 =

*a*1 (положение пика) =

*a*2 (ширина пика) =

Число итераций, потребовавшихся для аппроксимации =

Входные параметры: данные файла input из Практической работы № 8.

Библиографический список

1. *Белецки Я.* Фортран 77. М.: Высш. шк., 1991.
2. *Бухтияров А.М.*, *Маликова Ю.П.*, *Фролов Г.Д.* Практикум по программированию на Фортране (ОС ЕС ЭВМ). М.: Наука, 1979.
3. *Меткалф М.*, *Рид Дж.* Описание языка программирования ФОРТРАН 90. М.: Мир, 1995.
4. *Эберт К.*, *Эдерер Х.* Компьютеры. Применение в химии. М.: Мир, 1988.
5. *Мак-Кракен Д.*, *Дорн У.* Численные методы и программирование на Фортране. М.: Мир, 1977.
6. Портал «Tutorials point, Learn Fortran». URL: https://www.tutorialspoint.com/fortran/index.htm (дата обращения: 01.12.2019).
7. Портал «GCC, the GNU compiler collection». URL: https://gcc.gnu.org/onlinedocs/gcc-5.3.0/gfortran/ (дата обращения: 01.12.2019).
8. *Савельев И.В.* Курс физики: Учеб.: В 3 т. Т. 1: Механика. Молекулярная физика. М.: Наука, 1989.
9. *Некрасов Б.В.* Основы общей химии: [В 2 т.] Т. 1. СПб.: Лань, 2003.
10. *Лурье Ю.Ю.* Справочник по аналитической химии. М.: Химия, 1971.
11. Основы физической химии. Теория и задачи / В.В. Еремин и др. М.: Экзамен, 2005.
12. Сайт открытой программы Grace. URL: http://plasma-gate.weizmann.ac.il/Grace/ (дата обращения: 01.12.2019).
13. Вся высшая математика. Т. 6 / М.Л. Краснов и др. М.: Едиториал УРСС, 2003.
14. Физическая химия: В 2 кн. Кн. 1: Строение вещества. Термодинамика / К.С. Краснов и др. М.: Высш. шк., 2001.
15. *Debye P.* Zur Theorie der spezifischen Waerme // Annalen der Physik (in German). 1912. V. 39. P. 789–839.
16. *Стромберг А.Г.*, *Семченко Д.П.* Физическая химия. М.: Высш. шк., 2001.
17. Numerical recipes in Fortran 77: the art of scientific computing / William H. Press [et al.]. N. Y.: Cambridge University Press, 1992.
18. *Кудряшов И.В.*, *Каретников Г.С.* Сборник примеров и задач по физической химии. М.: Высш. шк., 1991.
19. Вся высшая математика. Т. 3 / М. Л. Краснов и др. М.: Едиториал УРСС, 2001.
20. *Ягодовский В.Д.* Статистическая термодинамика в физической химии. М.: БИНОМ. Лаборатория знаний, 2005.
21. Вся высшая математика. Т. 1 / М. Л. Краснов и др. М.: Едиториал УРСС, 2003.

22. Портал открытых исходных кодов библиотек линейной алгебры «Netlib». URL: www.netlib.org/lapack/explore-html (дата обращения: 01.12.2019).
23. *Marquardt D.W.* An Algorithm for least squares estimation of nonlinear parameters // Journal of the Society for Industrial and Applied Mathematics. 1963. V. 11. P. 431–441.
24. *Moré J.J.*, *Garbow B.S.*, *Hillstrom K.E.* User Guide for MINPACK-1. Argonne National Laboratory Report ANL-80–74, 1980.
25. Библиотека MINPACK. URL: https://people.sc.fsu.edu/~jburkardt/f_src/minpack/minpack.html (дата обращения: 01.12.2019).

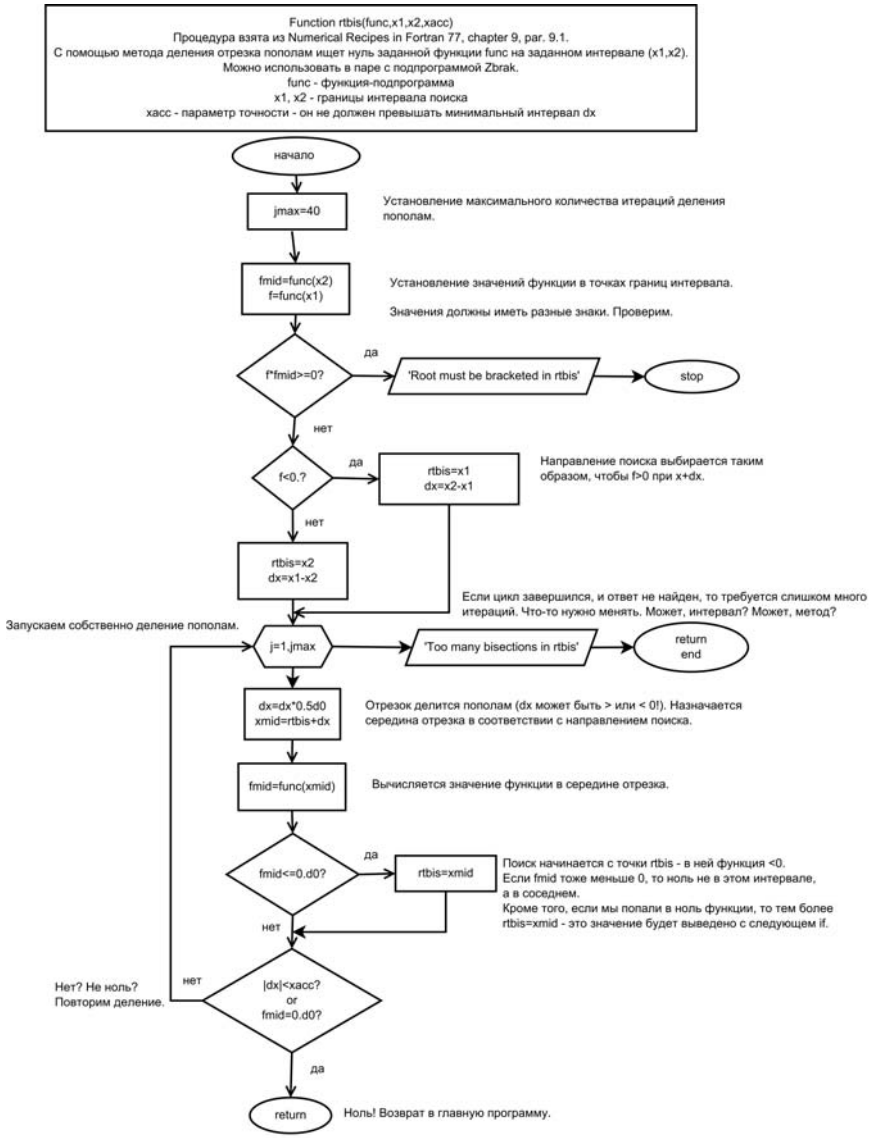
Дополнительная литература, рекомендуемая для чтения

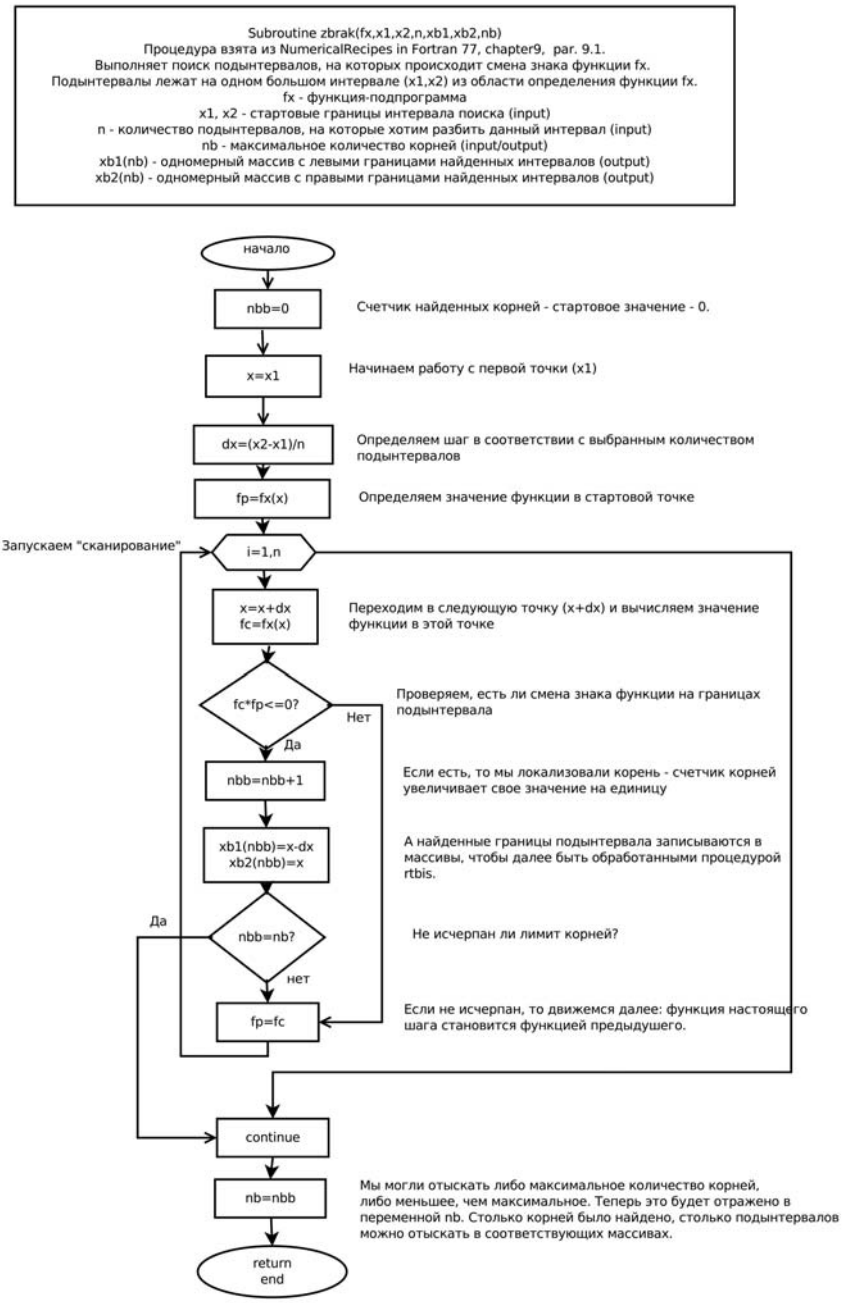
1. *Березин И.С.*, *Жидков Н.П.* Методы вычислений: В 2 т. М.: Государственное издательство физико-математической литературы, 1962.
2. *Берков Н.А.*, *Беркова Н.Н.* Алгоритмический язык Фортран 90. М.: МГИУ, 1998.
3. *Волков Е.А.* Численные методы. М.: Наука, 1987.
4. *Джонсон К.* Численные методы в химии. М.: Мир, 1983.
5. *Лапчик М.П.*, *Рагулина М.И.*, *Хеннер Е.К.* Численные методы. СПб.: Академия, 2009.
6. *Самарский А.А.* Введение в численные методы. СПб.: Лань, 2009.
7. *Срочко В.А.* Численные методы: Курс лекций. СПб.: Лань, 2010.
8. *Турчак Л.И.*, *Плотников П.В.* Основы численных методов. М.: Физматлит, 2003.

Приложения

Приложение 1

Блок-схемы подпрограмм ZBRAK и RTBIS для метода деления отрезка пополам. Составлены для одноименных подпрограмм, взятых из [17]




Subroutine zbrak(fx,x1,x2,n,xb1,xb2,nb)
Процедура взята из NumericalRecipes in Fortran 77, chapter9, par. 9.1.
Выполняет поиск подынтервалов, на которых происходит смена знака функции fx.
Подынтервалы лежат на одном большом интервале (x1,x2) из области определения функции fx.
fx - функция-подпрограмма
x1, x2 - стартовые границы интервала поиска (input)
n - количество подынтервалов, на которые хотим разбить данный интервал (input)
nb - максимальное количество корней (input/output)
xb1(nb) - одномерный массив с левыми границами найденных интервалов (output)
xb2(nb) - одномерный массив с правыми границами найденных интервалов (output)
начало
nbb=0
Счетчик найденных корней - стартовое значение - 0.
x=x1
Начинаем работу с первой точки (x1)
dx=(x2-x1)/n
Определяем шаг в соответствии с выбранным количеством подынтервалов
fp=fx(x)
Определяем значение функции в стартовой точке
Запускаем "сканирование"
i=1,n
x=x+dx
fc=fx(x)
Переходим в следующую точку (x+dx) и вычисляем значение функции в этой точке
fc*fp<=0?
Проверяем, есть ли смена знака функции на границах подынтервала
Нет
Да
nbb=nbb+1
Если есть, то мы локализовали корень - счетчик корней увеличивает свое значение на единицу
xb1(nbb)=x-dx
xb2(nbb)=x
А найденные границы подынтервала записываются в массивы, чтобы далее быть обработанными процедурой rtbis.
Да
nbb=nb?
Не исчерпан ли лимит корней?
нет
fp=fc
Если не исчерпан, то движемся далее: функция настоящего шага становится функцией предыдущего.
continue
nb=nbb
Мы могли отыскать либо максимальное количество корней, либо меньшее, чем максимальное. Теперь это будет отражено в переменной nb. Столько корней было найдено, столько подынтервалов можно отыскать в соответствующих массивах.
return
end

Приложение 2

Краткое описание метода Гаусса

Суть метода заключается в том, чтобы матрицу А путем эквивалентных преобразований привести к верхнетреугольному виду. Эквивалентных преобразований должно быть (N–1).

Итак, пусть дана система уравнений:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1N}x_N = b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2N}x_N = b_2, \\ a_{31}x_1 + a_{32}x_2 + \ldots + a_{3N}x_N = b_3 \\ \ldots \\ a_{N1}x_1 + a_{N2}x_2 + \ldots + a_{NN}x_N = b_N. \end{cases}$$

Прямой ход метода Гаусса

1-й шаг хода. На первом шаге основным уравнением для работы будет первое уравнение системы. Разделим его на a_{11}. Этот элемент называют *ведущим*.

$$x_1 + \frac{a_{12}}{a_{11}}x_2 + \ldots + \frac{a_{1N}}{a_{11}}x_N = \frac{b_1}{a_{11}}.$$

Теперь сначала умножим его на a_{21} и вычтем полученное уравнение из второго уравнения системы:

$$a_{21}x_1 + a_{21}\frac{a_{12}}{a_{11}}x_2 + \ldots + a_{21}\frac{a_{1N}}{a_{11}}x_N = a_{21}\frac{b_1}{a_{11}},$$

$$0 + \left(a_{22} - a_{21}\frac{a_{12}}{a_{11}}\right)x_2 + \ldots + \left(a_{2N} - a_{21}\frac{a_{1N}}{a_{11}}\right)x_N = \left(b_2 - a_{21}\frac{b_1}{a_{11}}\right).$$

Тогда получим видоизмененную систему:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1N}x_N = b_1, \\ 0 + a_{22}^{(1)}x_2 + \ldots + a_{2N}^{(1)}x_N = b_2^{(1)}, \\ a_{31}x_1 + a_{32}x_2 + \ldots + a_{3N}x_N = b_3, \\ \ldots \\ a_{N1}x_1 + a_{N2}x_2 + \ldots + a_{NN}x_N = b_N. \end{cases}$$

, где $a_{22}^{(1)} = \left(a_{22} - a_{21}\frac{a_{12}}{a_{11}} \right)$, $a_{2N}^{(1)} = \left(a_{2N} - a_{21}\frac{a_{1N}}{a_{11}} \right)$, $b_2^{(1)} = \left(b_2 - a_{21}\frac{b_1}{a_{11}} \right)$.

Процедуру повторим для третьего уравнения системы.

$$a_{31}x_1 + a_{31}\frac{a_{12}}{a_{11}}x_2 + \ldots + a_{31}\frac{a_{1N}}{a_{11}}x_N = a_{31}\frac{b_1}{a_{11}},$$

$$0 + \left(a_{32} - a_{31}\frac{a_{12}}{a_{11}} \right)x_2 + \ldots + \left(a_{3N} - a_{31}\frac{a_{1N}}{a_{11}} \right)x_N = \left(b_3 - a_{31}\frac{b_1}{a_{11}} \right),$$

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1N}x_N = b_1, \\ 0 + a_{22}^{(1)}x_2 + \ldots + a_{2N}^{(1)}x_N = b_2^{(1)}, \\ 0 + a_{32}^{(1)}x_2 + \ldots + a_{3N}^{(1)}x_N = b_3^{(1)}, \\ \ldots \\ a_{N1}x_1 + a_{N2}x_2 + \ldots + a_{NN}x_N = b_N. \end{cases}$$

И так далее, пока не получим систему, у которой весь первый столбец (за исключением первой строки) равен нулю:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1N}x_N = b_1, \\ 0 + a_{22}^{(1)}x_2 + \ldots + a_{2N}^{(1)}x_N = b_2^{(1)}, \\ 0 + a_{32}^{(1)}x_2 + \ldots + a_{3N}^{(1)}x_N = b_3^{(1)}, \\ \ldots \\ 0 + a_{N2}^{(1)}x_2 + \ldots + a_{NN}^{(1)}x_N = b_N^{(1)}. \end{cases}$$

2-й шаг метода. Теперь выбираем новый ведущий элемент – a_{22}. А основным уравнением для работы будет второе уравнение только что полученной системы.

$$x_2 + \frac{a_{23}^{(1)}}{a_{22}^{(1)}} x_3 + \ldots + \frac{a_{2N}^{(1)}}{a_{22}^{(1)}} x_N = \frac{b_2^{(1)}}{a_{22}^{(1)}},$$

$$a_{32}^{(1)} x_2 + a_{32}^{(1)} \frac{a_{23}^{(1)}}{a_{22}^{(1)}} x_3 + \ldots + a_{32}^{(1)} \frac{a_{2N}^{(1)}}{a_{22}^{(1)}} x_N = a_{32}^{(1)} \frac{b_2^{(1)}}{a_{22}^{(1)}},$$

$$0 + \left(a_{33}^{(1)} - a_{32}^{(1)} \frac{a_{23}^{(1)}}{a_{22}^{(1)}} \right) x_3 + \ldots + \left(a_{3N}^{(1)} - a_{32}^{(1)} \frac{a_{2N}^{(1)}}{a_{22}^{(1)}} \right) x_N = \left(b_3^{(1)} - a_{32}^{(1)} \frac{b_2^{(1)}}{a_{22}^{(1)}} \right).$$

Получаем систему

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1N}x_N = b_1, \\ 0 + a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \ldots + a_{2N}^{(1)}x_N = b_2^{(1)}, \\ 0 + 0 + a_{33}^{(2)}x_3 + \ldots + a_{3N}^{(2)}x_N = b_3^{(1)}, \\ \ldots \\ 0 + a_{N2}^{(1)}x_2 + a_{N3}^{(1)}x_3 + \ldots + a_{NN}^{(1)}x_N = b_N. \end{cases}$$

И наконец:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1N}x_N = b_1, \\ 0 + a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \ldots + a_{2N}^{(1)}x_N = b_2^{(1)}, \\ 0 + 0 + a_{33}^{(2)}x_3 + \ldots + a_{3N}^{(2)}x_N = b_3^{(1)}, \\ \ldots \\ 0 + 0 + a_{N3}^{(2)}x_3 + \ldots + a_{NN}^{(2)}x_N = b_N. \end{cases}$$

Шагов будет (N – 1). В самом конце прямого хода получаем верхнетреугольную матрицу:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 + \ldots + a_{1N}x_N = b_1, \\ 0 + a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + a_{24}^{(1)}x_4 + \ldots + a_{2N}^{(1)}x_N = b_2^{(1)}, \\ 0 + 0 + a_{33}^{(2)}x_3 + a_{34}^{(2)}x_4 + \ldots + a_{3N}^{(2)}x_N = b_3^{(2)}, \\ 0 + 0 + 0 + a_{44}^{(3)}x_4 + \ldots + a_{4N}^{(3)}x_N = b_4^{(3)}, \\ \qquad\qquad \ldots \qquad\qquad \ldots \qquad\qquad \ldots \\ 0 + 0 + 0 + 0 + \ldots + a_{NN}^{(N-1)}x_N = b_N^{(N-1)}. \end{cases}$$

Обратный ход подразумевает последовательную подстановку полученных значений x в предыдущие уравнения, начиная с последнего: $x_N = b_N^{(N-1)} / a_N^{(N-1)}$ – так находим все переменные.

Приложение 3

Описание подпрограмм LAPACK: DGETRF и DGETRS [22]

1. DGETRF (M, N, A, LDA, IPIV, INFO) – осуществляет LU-decomposition.

[in] M – количество строк матрицы A.

[in] N – количество столбцов матрицы A.

[in, out] A – двумерный массив размера (LDA, N).

На входе это матрица MxN, которую следует факторизовать, на выходе – составная матрица: ее верхний треугольник (включая диагональ) – матрица U, ее нижний треугольник (без диагонали) – матрица L.

Диагональные элементы матрицы L все полагаются равными единице. Сохранять их нет смысла.

[in] LDA – параметр массива A. Может быть меньше, чем число строк матрицы A, – это случай работы с подматрицей матрицы A. В нашем случае полагаем всегда LDA = M.

[out] IPIV – одномерный массив пересортировки длины либо M, либо N (в зависимости от того, что меньше). Содержит информацию о том, как были пересортированы строки матрицы A в процессе выбора ведущего элемента на каждом шаге решения системы уравнений. Это выходной параметр, без него невозможно расположить в нужном порядке решения уравнений, которые будут найдены позже в процедуре DGETRS.

[out] INFO – сигнал корректности работы процедуры. Возможно три значения на выходе:

= 0: корректная работа,

= -i: i-й аргумент имеет неверное значение,

= i: U (i, i) = 0. Факторизация была выполнена, но матрица U обладает особенностью.

2. DGETRS (TRANS, N, NRHS, A, LDA, IPIV, B, LDB, INFO) – решает систему линейных уравнений вида: A * X = B или A**T * X = B с участием матрицы общего вида A размерности NxN. Процедура использует факторизованную матрицу после LU-decomposition.

[in] TRANS – символьная переменная, определяет тип системы уравнений.

= 'N': A * X = B (матрица A не транспонируется).

= 'T': A**T* X = B (матрица A транспонируется).

[in] N – порядок матрицы A.

[in] NRHS – число столбцов матрицы B.

[in] A – составная матрица, полученная после факторизации. Размерность матрицы (LDA, N).

[in] LDA – параметр массива A. В нашем случае полагаем, что LDA = числу строк матрицы A. Как и в предыдущей процедуре.

[in] IPIV – одномерный массив пересортировки, полученный после факторизации.

[in, out] B – двумерный массив с параметрами (LDB, NRHS). На входе – матрица B (в нашем случае – это вектор свободных членов), на выходе – искомые переменные (вектор $\vec{x}$).

[in] LDB – параметр массива B. Положим LDB = числу строк матрицы B.

[out] INFO – сигнал корректности работы процедуры. Только два значения:

= 0: корректная работа,

= -i: i-й аргумент имеет неверное значение.

Приложение 4

Описание подпрограмм для кубической интерполяции: SPLINE и SPLINT

1. SPLINE (x, y, n, yp1, ypn, y2). Взята из [17].

Подпрограмма вычисляет массив вторых производных, необходимый для дальнейшей интерполяции. Работает в паре с процедурой SPLINT. Использует в своем составе трехдиагональный алгоритм на базе LU-decomposition для решения матричного уравнения AU = r. Размерность матрицы A – (n, n) – по числу табулированных точек исходной функции.

Параметры подпрограммы:

x, y – массивы интерполируемой функции (input). Значение x обязательно должно увеличиваться от точки к точке,

n – число точек этой функции (input),

yp1, ypn – первые производные в начальной и финальной точках (input).

Если указать yp1, ypn = 1.d31, то это будет соответствовать естественным граничным условиям.

y2 – массив вторых производных интерполируемой функции в табулированных точках (output).

2. SPLINT (xa, ya, y2a, n, x, y). Взята из [17].

Пользователь вводит значение точки x, для которой нужно вычислить интерполированное значение функции. Подпрограмма определяет интервал (x(j), x(j + 1)), в который попадает точка x, с помощью встроенного в программу метода *bisection*. Далее с помощью общей интерполяционной формулы вычисляет значение y в точке x. Константы a, в, c и d вычисляются по табулированным точкам, а вторые производные задаются массивом y2a (в программе SPLINE это был массив y2).

Параметры подпрограммы:

xa, ya – все те же массивы интерполируемой функции (input),

y2a – массив вторых производных, найденный в SPLINE (input),

n – количество точек исходной функции (input),

x – переменная, для которой нужно отыскать значение функции (input),

y – искомое (интерполированное) значение функции (output).

Учебное издание

Харлампиди Дарья Дмитриевна
Адамсон Сергей Олегович

ВЫЧИСЛИТЕЛЬНЫЕ МЕТОДЫ В ХИМИИ

Учебное пособие

Редактор *Алексеева А. А.*
Оформление обложки *Удовенко В. Г.*
Компьютерная верстка *Дорожкина О. Н., Ковтун М. А.*

Московский педагогический государственный университет (МПГУ).
119991, Москва, ул. Малая Пироговская, д. 1, стр. 1.

Управление издательской деятельности
и инновационного проектирования (УИД и ИП) МПГУ.
119571, Москва, пр-т Вернадского, д. 88, оф. 446,
тел. +7 (499) 730-38-61, e-mail: izdat@mpgu.su.

Отпечатано в отделе оперативной полиграфии
УИД и ИП МПГУ.

Подписано в печать 04.08.2020. Формат 60x90/16.
Бум. офсетная. Печать цифровая. Усл. печ. л. 5,0.
Тираж 500 экз. Заказ 1095.

ISBN 978-5-4263-0908-1