Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ
(ТУСУР)**

Кафедра автоматизированных систем управления (АСУ)

В. В. Романенко

ЧИСЛЕННЫЕ МЕТОДЫ

**Учебно-методическое пособие по лабораторным
работам и самостоятельной работе студентов**

Томск
2024

УДК 519.6
ББК 22.19
Р–69

Рецензент:
Исакова А. И., доцент кафедры автоматизированных систем управления ТУСУР, канд. техн. наук

Романенко, Владимир Васильевич
Р–69 Численные методы: учебно-методическое пособие по лабораторным работам и самостоятельной работе студентов / В. В. Романенко. – Томск : Томск. гос. ун-т систем упр. и радиоэлектроники, 2024. – 100 с.

Учебно-методическое пособие предназначено для студентов, изучающих дисциплину «Численные методы», и содержат все материалы, необходимые для выполнения лабораторных работ, а также организации самостоятельной работы: описание вычислительных алгоритмов и требуемого формата входных и выходных данных для программ, реализующих задания на лабораторные работы, образец титульного листа и рекомендуемую структуру отчета по лабораторным работам, а также дополнительные материалы, описывающие способы форматирования выходных данных в различных языках программирования, программной обработки заданных в аналитическом виде функций и методы работы с приближенными числами.

Одобрено на заседании кафедры автоматизированных систем управления (АСУ), протокол № 11 от 23.11.2023 г.

УДК 519.6
ББК 22.19

Оглавление

Введение

Численные методы – это методы, позволяющие при помощи алгоритмов, имеющих конечное число итераций, решать различные математические задачи (заданные в аналитическом виде). При этом набор инструкций, использующийся для написания алгоритма, ограничен и включает только такие инструкции, которые элементарно реализуются на ЭВМ (в данном случае, применительно к языкам высокого уровня). Таким образом, ограниченность набора инструкций и конечность алгоритма делает возможной его реализацию в виде программы. Решение же задач на ЭВМ в аналитическом виде затруднено.

Например, пусть нам требуется решить уравнение $f(x) = 0$ (т.е. найти *нули* функции). Очевидно, что аналитическое решение этого уравнения зависит от вида функции $f(x)$. Если это линейная функция, то уравнение решается одним методом, если это квадратный полином – другим. Существует множество методов для решения такого уравнения, если функция является полиномиальной, тригонометрической, экспоненциальной, содержит дифференциалы, интегралы и т.д. Однако, ЭВМ не может провести анализ функции (в чем и заключается смысл аналитического решения), и поэтому численные методы поиска нуля функции построены так, что не зависят от ее вида.

Численное решение априори является неточным, т.к. погрешности возникают как из-за использования приближенного алгоритма [1-2], так и по причине конечности разрядной сетки ЭВМ. Конечность разрядной сетки подразумевает, что не все числа ЭВМ может хранить без погрешно-

сти. Например, самый длинный тип данных, поддерживаемый математическим сопроцессором (FPU) и распространенными компиляторами, имеет размер 10 байт. При этом самое большое по модулю число, которое может уместиться в эти рамки, составляет $\sim 10^{4900}$, а самое маленькое – $\sim 10^{-4900}$. Но это не означает, что точность такого числа составляет 4900 знаков после запятой. Если у числа есть целая часть, то количество двоичных разрядов, остающихся для кодирования дробной части, уменьшается. Кроме того, в виду двоичности представления чисел в FPU, точно кодируются только те числа, которые являются целой степенью числа 2 (или суммой таких степеней). Например,

$$2_{10} = 10_2\ (1\cdot 2^1 + 0\cdot 2^0),$$

$$13_{10} = 1101_2\ (1\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0),$$

$$0.5_{10} = 0.1_2\ (1\cdot 2^{-1}),$$

$$0.3125_{10} = 0.0101_2\ (0\cdot 2^{-1} + 1\cdot 2^{-2} + 0\cdot 2^{-3} + 1\cdot 2^{-4}).$$

В противном случае число представляется в ЭВМ только с определенной погрешностью. Например, рассмотрим представление числа 0.1 с точностью до нескольких двоичных разрядов (количество разрядов указано в скобках):

$$0.1_{10} = 0.000_2 = 0_{10}\ (3);$$

$$0.1_{10} = 0.0001_2 = 0.063_{10}\ (4);$$

$$0.1_{10} = 0.00011_2 = 0.09375_{10}\ (5);$$

$$0.1_{10} = 0.000110_2 = 0.09375_{10}\ (6);$$

$$0.1_{10} = 0.0001100_2 = 0.09375_{10}\ (7);$$

$$0.1_{10} = 0.00011001_2 = 0.09765625_{10}\ (8)\ \text{и т.д.}$$

Как видно, точно это число представить двоичными разрядами невозможно. К сказанному можно еще добавить,

что дополнительные погрешности появляются при вычислении блоком FPU различных функций – тригонометрических, логарифмических, степенных и т.п. Все эти функции вычисляются либо при помощи каких-либо алгоритмов (разложение в ряды и т.д.), либо при помощи интерполяции по табличным значениям. В первом случае погрешность возникает по причине неточности алгоритма, невозможности вычисления бесконечного ряда и т.п. Во втором случае возникает погрешность интерполяции, поэтому и значение функции получается неточным.

Следует также отметить, что не все задачи имеют аналитическое решение. Например, известно, что существуют неберущиеся интегралы. Если такой интеграл входит в интегральное уравнение, то решить его аналитически не удастся. Можно привести и более простые примеры функций, найти нули которых аналитическим способом (в общем виде) невозможно:

$$f(x) = ax^3 + bx^2 + cx + d,$$

$$f(x) = ax + \cos(bx).$$

Здесь a, b, c, d – произвольные константы.

Проведение сложных математических расчетов требуется во многих отраслях науки и техники. При этом объем этих расчетов таков, что вручную за разумное время их выполнить невозможно. Примеры – распределение нагрузки между подключенными к электростанции объектами (оно должно происходить практически мгновенно при изменении потребляемой мощности), вычисление траектории космических тел, расчет движений земной коры в геоинформационных системах (а это задачи нефтяной, газовой и других

отраслей) и многое другое. Для этого и внедряются в промышленность и науку вычислительные системы и пишутся специализированные пакеты для проведения численных расчетов. Распространение же ЭВМ ставит, в свою очередь, новые математические задачи, не существовавшие ранее – распределение Internet-трафика, обсчет трехмерных моделей в графических редакторах и играх, машинное обучение искусственных нейронных сетей, навигация и управление беспилотными средствами (автомобилями, БПЛА) и т.д.

Таким образом, знание численных методов необходимо инженеру, область деятельности которого связана с программным обеспечением вычислительной техники и, в особенности, автоматизированных систем.

В рамках курса «Численные методы» необходимо выполнить следующие лабораторные работы:

1. Решение уравнений с одной переменной.
2. Решение задач линейной алгебры.
3. Вычисление собственных чисел и собственных векторов.
4. Решение систем нелинейных уравнений.
5. Интерполирование и численное дифференцирование функций.
6. Приближение сплайнами.
7. Численное интегрирование функций.
8. Решение обыкновенных дифференциальных уравнений.
9. Решение интегральных уравнений 1-го и 2-го рода.

Как видно, рассматриваемые задачи принадлежат к двум большим классам: численное решение уравнений или

систем уравнений (темы 1-4, 8-9) и приближение функций (темы 5-7).

Обязательными для выполнения являются шесть работ – №1, №2, №3, №5, №6 и №7. Остальные темы (№4, №8 и №9) изучаются в рамках самостоятельной работы, а соответствующие лабораторные работы выполняются по желанию.

1 Решение уравнений с одной переменной

В ходе данной лабораторной работы необходимо реализовать ряд методов решения уравнений

$$f(x) = 0, \tag{1.1}$$

где $x \in [a, b]$ – скалярный аргумент функции f. При этом предполагается, что отделение корней уже произведено, т.е. на отрезке $[a, b]$ находится только одно решение уравнения (1.1) $\xi \in [a, b]$, или, другими словами, только один нуль функции $f(x)$, т.е. $f(\xi) \equiv 0$. В этом случае выполняется условие

$$f(a)\, f(b) \le 0. \tag{1.2}$$

Решение должно быть найдено с абсолютной погрешностью по аргументу ε и/или абсолютной погрешностью по значению функции δ, т.е.

$$|\xi - x^*| < \varepsilon \text{ и/или} \tag{1.3}$$

$$|f(x^*)| < \delta, \tag{1.4}$$

где ξ – точное решение уравнения (1.1), а x^* – приближенное.

Зачем использовать две различные погрешности? Дело в том, что, в зависимости от вида функции, погрешность решения по аргументу и по значению функции могут не совпадать. Например, рассмотрим быстро растущую функцию. Из рис. 1.1 видно, что даже если по аргументу требуемая точность решения достигнута, то по значению функции – нет. Такая же ситуация будет наблюдаться для быстро убывающей функции (т.е. для любой функции, имеющей на исследуемом отрезке большую производную).

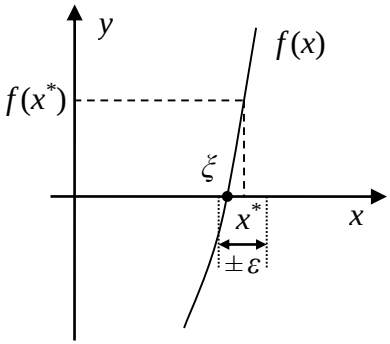

Рисунок 1.1 – Пример функции с большим (по модулю) значением производной вблизи корня

Обратная ситуация будет наблюдаться для функции с малыми значениями производной – при достижении требуемой точности по значению функции, точность по аргументу достигнута не будет (рис. 1.2).

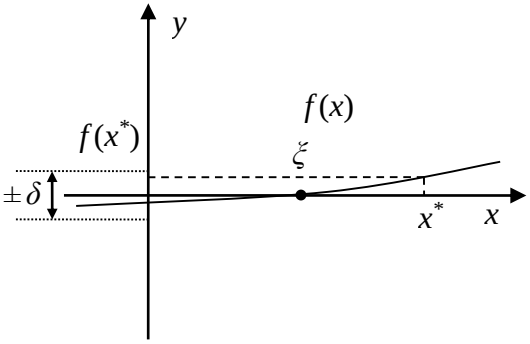

Рисунок 1.2 – Пример функции с малым (по модулю) значением производной вблизи корня

Для упрощения можно положить $\varepsilon = \delta$. Так как точный корень нам неизвестен, то условие (1.3) в численных методах заменяют другими, альтернативными, условиями, которые мы рассмотрим ниже.

1.1 Методы решения

Для решения уравнения (1.1) необходимо реализовать три обязательных метода (дихотомии, хорд и Ньютона) и, по желанию, три дополнительных (комбинированный метод, метод итераций и золотого сечения).

1.1.1 Интервальные методы

Методы дихотомии, хорд и золотого сечения являются интервальными, т.е. их смысл заключается в уменьшении исходного интервала, содержащего корень, до тех пор, пока размеры интервала не окажутся соизмеримы с требуемой погрешностью.

Для этих методов интервалом поиска корня на некоторой k-й итерации будет являться отрезок $[a_k, b_k]$, при этом $a_0 = a$, $b_0 = b$. Длина интервала в интервальных методах гарантированно уменьшается на каждой итерации решения, поэтому альтернативой условию (1.3) будет, очевидно, условие

$$\frac{b_k - a_k}{2} < \varepsilon, \tag{1.5}$$

т.к. погрешность определения корня не может превышать половины длины интервала.

В методе дихотомии интервал разбивается следующим образом. Вычисляется точка, расположенная в середине отрезка:

$$c_k = \frac{a_k + b_k}{2}. \tag{1.6}$$

Далее, согласно (1.2), проверяется, какому из интервалов – $[a_k, c_k]$ или $[c_k, b_k]$ – принадлежит корень. Т.е.,

$$\begin{cases} \text{Если } \ f(a_k)\, f(c_k) \le 0 \to a_{k+1} = a_k,\ b_{k+1} = c_k, \\ \text{иначе} \to a_{k+1} = c_k,\ b_{k+1} = b_k. \end{cases} \qquad (1.7)$$

В качестве *k*-го приближения корня берется точка

$$x_k = \frac{a_k + b_k}{2}. \qquad (1.8)$$

В методе хорд интервал разбивается другой точкой:

$$c_k = a_k - \frac{f(a_k)}{f(b_k) - f(a_k)}(b_k - a_k). \qquad (1.9)$$

Выбор интервала осуществляется согласно (1.7), а новое приближение корня совпадает с точкой c_k ($x_k = c_k$). Однако, в отличие от других интервальных методов, в методе хорд постоянное уменьшение длины интервала не гарантировано, поэтому погрешность рассчитывается по формуле итерационных методов (1.13).

В методе золотого сечения интервал разбивается двумя симметричными относительно границ интервала точками:

$$d_k = a_k + \frac{b_k - a_k}{\gamma},\ c_k = a_k + \frac{b_k - a_k}{\gamma^2}, \qquad (1.10)$$

где $\gamma = \dfrac{\sqrt{5}+1}{2}$.

Для упрощения вычислений можно учесть упомянутую симметричность расположения точек c_k и d_k:

$$c_k - a_k = b_k - d_k. \qquad (1.11)$$

Далее, согласно (1.2), проверяется, какому из интервалов – [a_k, d_k] или [c_k, b_k] – принадлежит корень. Т.е.,

$$\begin{cases} \text{Если } \ f(a_k)\, f(d_k) \le 0 \to a_{k+1} = a_k,\ b_{k+1} = d_k, \\ \text{иначе} \to a_{k+1} = c_k,\ b_{k+1} = b_k. \end{cases} \qquad (1.12)$$

Новое приближение корня вычисляется по формуле (1.8).

1.1.2 Итерационные методы

Методы Ньютона (касательных) и итераций являются итеративными (итерационными), на основе некоторого приближения корня x_k они позволяют на каждой итерации получать новое приближение x_{k+1}. При этом используется информация о первой производной функции. Вместо условия (1.3) в итеративных методах оценивается расстояние между последним и предпоследним приближениями корня:

$$|x_{k+1} - x_k| < \varepsilon. \tag{1.13}$$

При этом нужно знать начальное приближение x_0, а дальнейшие приближения на каждой k+1-й итерации находятся по итеративной формуле:

$$x_{k+1} = \varphi(x_k). \tag{1.14}$$

В методе Ньютона начальное приближение выбирается в соответствии со следующим условием: если в некоторой точке x произведение $f(x)f''(x) > 0$, то точка x является подходящей для начала итерационного процесса. Проверяются границы интервала:

$$\begin{cases} \text{Если } f(a)f''(a) > 0, \text{ то } x_0 = a, \\ \text{Если } f(b)f''(b) > 0, \text{ то } x_0 = b. \end{cases} \tag{1.15}$$

На практике может наблюдаться ситуация, когда оба условия (1.15) не выполняются. В этом случае вместо второго условия можно использовать оператор «иначе», либо воспользоваться вторым критерием.

Если вторая производная функции не известна, можно воспользоваться другим критерием. Вычислим точку c по формуле (1.9), и далее

$$\begin{cases} \text{Если } f(a)\,f(c) \le 0, \text{ то } x_0 = a; \\ \text{иначе } x_0 = b. \end{cases} \tag{1.16}$$

Если начальная точка определена неправильно, то найденное решение уравнения (1.1) может находиться за пределами отрезка [*a, b*].

Функция $\varphi(x_k)$ в (1.14) для метода Ньютона выглядит следующим образом:

$$\varphi(x_k) = x_k - \frac{f(x_k)}{f'(x_k)}. \tag{1.17}$$

В методе итераций, если выполняется неравенство $|\varphi'(x)| < 1$, процесс сходится независимо от выбора начальной точки. Поэтому можно брать любую из границ интервала, его середину и т.п. А функция $\varphi(x_k)$ в (1.14) выглядит следующим образом:

$$\varphi(x_k) = x_k - \frac{f(x_k)}{\max\limits_{[a,b]} |f'(x)|}. \tag{1.18}$$

В отличие от интервальных методов, длина исследуемого отрезка в которых на каждой итерации гарантированно уменьшается (например, для метода дихотомии – в два раза, для метода золотого сечения – в γ раз), в итеративных методах, в общем случае, расстояние между последовательными приближениями корня может иногда и увеличиваться. То же самое касается и значения функции в этих точках – оно может как уменьшаться, так и увеличиваться. Поэтому для некоторых функций условия (1.3) и (1.4) могут не выполняться в течение довольно большого числа итераций (или вообще никогда). В этом случае итерации следует прекращать при выполнении хотя бы одного условия.

1.1.2 Комбинированный метод

Комбинированный метод сочетает в себе сильные стороны методов хорд и Ньютона, и поэтому является достаточно эффективным для большого класса функций. Т.к. он является интервальным, то для него применимы выражения (1.5) и (1.8). Исключение интервалов выполняется по следующему алгоритму.

Сначала по формуле (1.9) ищется точка пересечения хорды с осью *x*. Далее, согласно (1.15), если $f(a_k)f''(a_k) > 0$ то точку a_k можно переместить ближе к корню по формуле Ньютона (1.14) и (1.17). Тогда точка b_k перемещается по формуле метода хорд (1.9):

$$\begin{cases} a_{k+1} = a_k - \dfrac{f(a_k)}{f'(a_k)}, \\ b_{k+1} = c_k. \end{cases} \tag{1.19}$$

Если же $f(b_k)f''(b_k) > 0$, то, наоборот, точку b_k можно переместить ближе к корню по формуле Ньютона, а точку a_k – по формуле метода хорд:

$$\begin{cases} a_{k+1} = c_k, \\ b_{k+1} = b_k - \dfrac{f(b_k)}{f'(b_k)}. \end{cases} \tag{1.20}$$

Два упомянутых условия достаточно проверять только один раз, если вторая производная не меняет своего знака на отрезке [*a*, *b*]. Но, т.к. это выполняется не для всех функций, лучше их проверять на каждой итерации. Аналогично (1.15), вместо второго условия можно использовать оператор «иначе», чтобы не возникла ситуация, когда оба условия не выполняются.

1.2 Формат входных данных

Формат входного файла:

`n` – номер метода (в порядке их перечисления в п. 1.1, т.е. 1 – дихотомии, 2 – хорд и т.д.);

`f(x)` – исследуемая функция в аналитическом виде;

`a b` – границы отрезка;

`ε` – требуемая точность решения.

1.3 Формат выходных данных

Формат выходного файла:

x^* – решение уравнения;

$f(x^*)$ – значение функции в найденной точке x^*;

ε^* – погрешность полученного решения.

2 Решение задач линейной алгебры

К решению задач линейной алгебры в численных методах относят решение систем линейных алгебраических уравнений (СЛАУ) и вычисление различных характеристик матриц – определителей, обратных матриц, собственных чисел и собственных векторов. Для равномерного распределения нагрузки, вычисление собственных чисел и собственных векторов матриц вынесено в отдельную лабораторную работу (№3).

К решению систем линейных уравнений сводятся многие задачи автоматизации. Например, распределение нагрузки на электростанции, о которой упоминалось во введении. Порядок матриц в таких задачах достигает огромных величин. Также при помощи матриц выполняются различные операции над многомерными объектами (в физике, компьютерной графике и т.п.). Матричные преобразования играют большую роль при написании программного обеспечения многопроцессорных ЭВМ (т.н. параллельное программирование, ПП). Учитывая распространенность многопроцессорных и многоядерных ПК в настоящее время (а также *кластеров* из таких ПК), специалисты в области ПП становятся все более востребованными.

Все перечисленные характеристики матриц, так или иначе, находятся при помощи решения некоторых СЛАУ. СЛАУ выглядит следующим образом:

$$Ax = b, \tag{2.1}$$

где A – матрица размером $n \times m$, x – вектор неизвестных длиной m, b – вектор свободных коэффициентов длиной n. Все векторы являются столбцами.

Если $n < m$, то СЛАУ называется недоопределенной, а если $n > m$ – то переопределенной. Мы будем рассматривать только нормально определенные системы с $n = m$ (т.е. имеющие квадратную матрицу A).

Точность решения СЛАУ можно оценить, вычислив вектор невязки:

$$\varepsilon = Ax^* - b, \tag{2.2}$$

где x^* – приближенное решение СЛАУ.

Для получения скалярной оценки можно использовать норму (1.4).

Учитывая, что точное решение уравнения (2.1) для квадратной матрицы можно найти аналитически, т.е.

$$x^* = A^{-1}b, \tag{2.3}$$

можно сделать вывод, что единственное решение существует только тогда, когда существует обратная матрица. А для этого, в свою очередь, требуется, чтобы

$$\det A \neq 0. \tag{2.4}$$

2.1 Методы решения

Существуют три класса методов решения СЛАУ [2]:

1. Прямые (точные). Дают решение задачи за конечное число итераций, при этом, если все операции выполняются точно, то и решение получается точным. При реализации на ЭВМ погрешность, конечно же, появляется (по описанным выше причинам – конечность разрядной сетки и т.д.). К прямым методам относятся методы Гаусса, декомпозиции (Халецкого), ортогонализации и др. Прямые методы применяются для решения систем порядка 10^3.

2. Итерационные. Дают решение с некоторой точностью как предел последовательных приближений. К итерационным методам относятся методы релаксации, простой итерации, Зейделя, градиентные методы и др. Итерационные методы применяются для систем порядка 10^7.

3. Вероятностные. Основаны на случайных испытаниях некоторой блуждающей частицы, моделирующей решение задачи и применении закона больших чисел. В основном, это метод Монте-Карло и его модификации.

В данной лабораторной работе необходимо реализовать один из трех обязательных точных методов (в зависимости от номера варианта):

1) метод Гаусса;
2) метод декомпозиции;
3) метод ортогонализации (схема №1).

Дополнительно можно реализовать еще один итерационный метод – Зейделя или простой итерации.

При помощи данных методов необходимо реализовать решение следующих задач:

1) решение СЛАУ;
2) поиск определителя матрицы (только для методов Гаусса и декомпозиции).
3) вычисление обратной матрицы.

2.1.1 Метод Гаусса

Прямой ход (преобразование матрицы к треугольному виду):

$$a_{kk}^{(k)} = 1,\ a_{kj}^{(k)} = \frac{a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}},\ b_k^{(k)} = \frac{b_k^{(k-1)}}{a_{kk}^{(k-1)}}; \qquad (2.5)$$

$$a_{ik}^{(k)} = 0,\ a_{ij}^{(k)} = a_{ij}^{(k-1)} - a_{ik}^{(k-1)} a_{kj}^{(k)},$$
$$b_i^{(k)} = b_i^{(k-1)} - a_{ik}^{(k-1)} b_k^{(k)}. \qquad (2.6)$$

Здесь $k = 1, 2, \ldots, n$, $i = k + 1, k + 2, \ldots, n$, $j = k + 1, k + 2, \ldots, n$. Значения 1 и 0, которые используются в виде констант, можно получить и по общим формулам, но это приведет к ненужным погрешностям. После прямого хода СЛАУ примет следующий вид:

$$\begin{pmatrix} 1 & a_{12}^{(1)} & \ldots & a_{1n}^{(1)} \\ 0 & 1 & \ldots & a_{2n}^{(2)} \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & 1 \end{pmatrix} x = \begin{pmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \ldots \\ b_n^{(n)} \end{pmatrix}. \qquad (2.7)$$

Из анализа (2.7) очевидны формулы для обратного хода (получения решения СЛАУ):

$$x_i = b_i^{(i)} - \sum_{j=i+1}^{n} a_{ij}^{(i)} x_j,\ i = n, n-1, \ldots, 1. \qquad (2.8)$$

Определитель исходной матрицы A можно вычислить по формуле

$$\det A = \prod_{i=1}^{n} a_{ii}^{(i-1)} = a_{11} \cdot a_{22}^{(1)} \cdot \ldots \cdot a_{nn}^{(n-1)}. \qquad (2.9)$$

Во всех формулах подразумевается, что $a_{ij}^{(0)} = a_{ij}$.

Метод Гаусса обладает следующим недостатком. Если обратить внимание на формулу (2.5), то видно, что в ней происходит операция деления на диагональные элементы матриц $A^{(k)}$. Если в процессе решения требуемый диагональный элемент получится равным нулю, то этот метод даст сбой, даже если условие (2.4) выполняется. В этом случае требуется перестановка строк исходной матрицы A (и соответствующих элементов вектора b). В данной лабораторной

работе делать этого не требуется, хотя программная реализация достаточно простая.

2.1.2 Метод декомпозиции

Сначала исходная матрица A раскладывается на две треугольные матрицы B и C таким образом, что $A = BC$. Формулы для получения элементов матриц B и C:

$$b_{ij} = a_{ij} - \sum_{k=1}^{j-1} b_{ik} c_{kj}, \quad j = 1, 2, ..., n, \quad i = j, j+1, ..., n; \tag{2.10}$$

$$c_{ij} = \frac{1}{b_{ii}} \left(a_{ij} - \sum_{k=1}^{i-1} b_{ik} c_{kj} \right), \quad i = 1, 2, ..., n-1, \quad j = i+1, i+2, ..., n. \tag{2.11}$$

Диагональные элементы матрицы C равны 1, остальные элементы матриц B и C нулевые:

$$B = \begin{pmatrix} b_{11} & 0 & ... & 0 \\ b_{21} & b_{22} & ... & 0 \\ ... & ... & ... & ... \\ b_{n1} & b_{n2} & ... & b_{nn} \end{pmatrix}, \quad C = \begin{pmatrix} 1 & c_{12} & ... & c_{1n} \\ 0 & 1 & ... & c_{2n} \\ ... & ... & ... & ... \\ 0 & 0 & ... & 1 \end{pmatrix}.$$

Важен порядок вычисления элементов матриц B и C. Сначала вычисляется первый столбец матрицы B, затем первая строка матрицы C, затем второй столбец B, затем вторая строка C и т.д.

После этого сначала решается СЛАУ $By = d$, а затем – $Cx = y$. По аналогии с (2.8), для решения этих систем можно записать

$$y_i = \frac{1}{b_{ii}} \left(b_i - \sum_{k=1}^{i-1} b_{ik} y_k \right), \quad i = 1, 2, ..., n; \tag{2.12}$$

$$x_i = y_i - \sum_{k=i+1}^{n} c_{ik} x_k, \ \ i = n, n-1, ..., 1. \qquad (2.13)$$

Определитель исходной матрицы A можно вычислить по формуле

$$\det A = \det BC = \det B \cdot \det C = \prod_{i=1}^{n} b_{ii}. \qquad (2.14)$$

Метод декомпозиции обладает тем же недостатком, что и метод Гаусса. В формуле (2.11) происходит деление на диагональные элементы матрицы B. Если в процессе решения требуемый диагональный элемент получится равным нулю, то этот метод также даст сбой. Аналогично, тогда может помочь только перестановка строк исходной СЛАУ, но делать это в рамках данной лабораторной работы не требуется.

2.1.3 Метод ортогонализации

Метод ортогонализации лишен этого недостатка. Как видно из формул, приведенных ниже, деление на ноль он может дать только в том случае, если одна из строк матрицы U будет содержать только нули. Но при выполнении условия (2.4) это невозможно.

Итак, сначала исходная матрица A преобразуется в расширенную матрицу A' размера $(n+1)\times(n+1)$:

$$\begin{gathered} a'_{ij} = a_{ij}, \ \ a'_{i,n+1} = -b_i, \ \ i = 1, 2, ..., n, \ \ j = 1, 2, ..., n; \\ a'_{n+1,j} = e_{n+1,j}, \ \ j = 1, 2, ..., n+1. \end{gathered} \qquad (2.15)$$

Здесь e_{n+1} – $n+1$-я строка единичной матрицы. Расширенный вектор x' дополняется еще одним компонентом, и его размер для расширенной системы составляет $n+1$. Сама же расширенная СЛАУ будет выглядеть так:

$$A' x' = 0. \qquad (2.16)$$

Чтобы расширенная система была эквивалентна исходной, последний компонент вектора x' должен быть равен единице, т.е.

$$x'_j = x_j,\ \ x'_{n+1} = 1,\ \ j = 1, 2, ..., n. \tag{2.17}$$

Таким образом, имеем следующую расширенную систему:

$$\begin{pmatrix} a_{11} & a_{12} & ... & -b_1 \\ a_{21} & a_{22} & ... & -b_2 \\ ... & ... & ... & ... \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ ... \\ 1 \end{pmatrix} = 0.$$

Далее последовательно находятся строки некоторых матриц U и Z. Их размер также составляет $(n+1)\times(n+1)$:

$$u_i = a_i - \sum_{j=1}^{i-1} \left(a_i, z_j\right) z_j,\ \ z_i = \frac{u_i}{\|u_i\|}, \tag{2.18}$$
$$i = 1, 2, ..., n+1.$$

Здесь a_i, u_i, z_i – соответствующие строки матриц A', U и Z. В скобках стоит скалярное произведение, а норма в данном случае – это квадратный корень из скалярного произведения вектора самого на себя, т.е. может быть вычислена по формуле (1.4).

После этого можем получить решение СЛАУ:

$$x_i = \frac{z_{n+1,i}}{z_{n+1,n+1}},\ \ i = 1, 2, ..., n. \tag{2.19}$$

Очевидно, что при программировании можно обойтись всего одной матрицей:

$$a_i = \frac{\alpha}{\|\alpha\|},\ \text{где}\ \ \alpha = a_i - \sum_{j=1}^{i-1} \left(a_i, a_j\right) a_j.$$

2.1.4 Метод простой итерации

Преобразуем исходную систему к виду

$$x = \beta + \alpha x, \tag{2.20}$$

где α – матрица размера $n \times n$, β – вектор размера n:

$$\beta_i = \frac{b_i}{a_{ii}}, \ i = 1, 2, ..., n,$$
$$\alpha_{ij} = -\frac{a_{ij}}{a_{ii}}, \ \alpha_{ii} = 0, \ j = 1, 2, ..., n, \ j \neq i. \tag{2.21}$$

Полагая в качестве начального приближения решения $x^{(0)} = \beta$, строим итерационный процесс по формулам

$$x^{(k+1)} = \beta + \alpha x^{(k)}. \tag{2.22}$$

Итерации заканчиваются, когда выполняется условие

$$\left\| x^{(k+1)} - x^{(k)} \right\| < \frac{1 - \|\alpha\|}{\|\alpha\|} \varepsilon, \tag{2.23}$$

где ε – требуемая точность решения.

Из (2.21) следует, что диагональные элементы исходной матрицы должны быть ненулевыми. Более того, на самом деле требования к ним еще жестче. Итерационный процесс (2.22) сходится, если норма матрицы α меньше 1. Для этого требуется, чтобы у исходной матрицы СЛАУ A числа, стоящие на главной диагонали, были больше суммы остальных чисел в соответствующей строке матрицы (все числа нужно брать по модулю), т.е.

$$|a_{ii}| > \sum_{\substack{j=1, \\ j \neq i}}^{n} |a_{ij}|, \ i = 1, 2, ..., n. \tag{2.24}$$

2.1.5 Метод Зейделя

Метод Зейделя является модификацией метода простой итерации. Поэтому преобразование (2.20), (2.21), а также критерий останова (2.23) верны и для него. Несколько по-другому строится итерационный процесс:

$$x_i^{(k+1)} = \beta_i + \sum_{j=1}^{i-1} \alpha_{ij} x_j^{(k+1)} + \sum_{j=i}^{n} \alpha_{ij} x_j^{(k)}. \qquad (2.25)$$

Ограничение (2.24) также применимо. Но, в силу модификаций, метод Зейделя сходится также для любой СЛАУ с симметричной положительно определенной матрицей. Чтобы сделать матрицу таковой, необходимо ее транспонировать и умножить на саму себя. Тогда аналогичные преобразования необходимо проделать и с правой частью СЛАУ:

$$A^{\mathrm{T}}Ax = A^{\mathrm{T}}b. \qquad (2.26)$$

Получаем систему $A'x = b'$, которую решаем методом Зейделя.

2.1.6 Вычисление обратных матриц

Обозначим как X неизвестные элементы обратной матрицы. Следовательно, нам необходимо решить систему

$$AX = E, \qquad (2.27)$$

где E – единичная матрица, т.е.

$$E = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{pmatrix}.$$

Все матрицы имеют размер $n \times n$. Решение матричной системы (2.27) можно представить в виде решения n СЛАУ

$$Ax_i = e_i,\ i = 1, 2, \ldots, n, \qquad (2.28)$$

где x_i, e_i – i-й столбец обратной и единичной матрицы соответственно.

Обратите внимание, что при вычислении обратной матрицы СЛАУ решается n раз, где n – порядок матрицы. При этом все треугольные матрицы в методах Гаусса и декомпозиции получаются одинаковыми, меняется только вектор свободных коэффициентов. Это нужно использовать для оптимизации вычислений в программе – все треугольные матрицы должны вычисляться только один раз.

2.2 Формат входных данных

Формат входного файла:

`m` – тип задачи (в том порядке, в котором они перечислены выше);

`n` – порядок матрицы;

`a11…a1n [b1]`
`a21…a2n [b2]`
`……`
`an1…ann [bn]`

– коэффициенты матрицы и вектор свободных коэффициентов (при решении СЛАУ, т.е. при `m` = 1).

2.3 Формат выходных данных

Формат выходного файла зависит от метода и типа задачи:

- Если используется метод Гаусса, то в любом случае в выходной файл выводятся матрицы $A^{(1)}$, $A^{(2)}$, …, $A^{(n)}$. Если решалась система СЛАУ, то еще и векторы $b^{(1)}$, $b^{(2)}$, …, $b^{(n)}$. Если вычислялась обратная матрица – векторы $e_1^{(n)}$, $e_2^{(n)}$, …, $e_n^{(n)}$.

- Если используется метод декомпозиции, то в любом случае выводятся матрицы B и C. Если решалась система СЛАУ, то вектор y. Если вычислялась обратная матрица – векторы y_1, y_2, …, y_n.
- Если используется метод ортогонализации, то в любом случае выводится расширенная матрица A'. При решении СЛАУ выводятся матрицы U и Z. Если вычислялась обратная матрица – матрицы U_1, Z_1, U_2, Z_2, …, U_n, Z_n.
- Для итерационных методов выводятся матрицы α и векторы β (для каждой решаемой СЛАУ).

При решении СЛАУ в файл выводятся:

x^* – вектор решения;

ε – вектор невязки;

$\|\varepsilon\|$ – норма вектора невязки.

При поиске определителя – его значение. При вычислении обратной матрицы – следующие величины:

X – обратная матрица;

ε – матрица невязки ($AX - E$);

$\|\varepsilon\|$ – норма матрицы невязки.

3 Вычисление собственных чисел и собственных векторов

Собственные числа и векторы квадратной матрицы являются ее важными характеристиками, использующимися в различных формах математического анализа. Собственное число матрицы λ_i и соответствующий ему собственный вектор x_i удовлетворяют следующему соотношению:

$$Ax_i = \lambda_i x_i. \tag{3.1}$$

У квадратной матрицы размерности n имеется n собственных чисел и векторов. Некоторые из них могут быть кратными (т.е. совпадающими). Таким образом, квадратная матрица размерности n имеет m различных собственных чисел λ_i и соответствующих им собственных векторов x_i кратности k_i. При этом

$$\sum_{i=1}^{m} k_i = n,\ \ i = 1, 2, ..., n,\ \ 1 \le m \le n. \tag{3.2}$$

Отметим также, что от умножения собственного вектора матрицы на скаляр c он не перестает быть ее собственным вектором:

$$A(cx_i) = \lambda_i(cx_i) \Rightarrow cAx_i = c\lambda_i x_i \Rightarrow Ax_i = \lambda_i x_i. \tag{3.3}$$

3.1 Методы решения

При аналитическом решении собственные числа матрицы находятся из решения уравнения

$$D(\lambda) = 0, \tag{3.4}$$

где $D(\lambda) = \det(A - \lambda E)$ – характеристический полином матрицы. После этого, согласно (3.1), можно найти собственные векторы, решая СЛАУ

$$(A - \lambda E)x = 0. \tag{3.5}$$

3.1.1 Вычисление собственных чисел методом Данилевского

В данной лабораторной работе для поиска собственных чисел и векторов мы будем использовать метод Данилевского. Суть его состоит в том, что исходная матрица A преобразуется в подобную ей матрицу Фробениуса P, имеющую следующий вид:

$$P = \begin{pmatrix} p_1 & p_2 & \cdots & p_{n-1} & p_n \\ 1 & 0 & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Делается это при помощи следующего преобразования подобия:

$$P = S^{-1}AS, \tag{3.6}$$

где $S = M_{n-1}M_{n-2}\ldots M_1,\ S^{-1} = M_1^{-1}M_2^{-1}\ldots M_{n-1}^{-1}$.

Таким образом, можно последовательно находить n–1 матрицу $A^{(k)}$:

$$\begin{gathered} A^{(k)} = M_{n-k}^{-1}A^{(k-1)}M_{n-k},\ k = 1,2,\ldots,n-1, \\ A^{(0)} = A,\ P = A^{(n-1)}. \end{gathered} \tag{3.7}$$

А можно найти матрицы S (прямую и обратную) и затем сразу вычислить P по формуле (3.6). Такой способ эффективнее, т.к. не нужно хранить множество матриц M, произведение которых еще понадобятся для вычисления собственных векторов.

Матрицы M строятся следующим образом:

$$M_k: \begin{cases} m_{ij} = e_{ij}, \ i = 1,2,...,n, \ \ j = 1,2,...,n, \ \ i \neq k; \\ m_{kj} = -\dfrac{a_{k+1,j}^{(n-k-1)}}{a_{k+1,k}^{(n-k-1)}}, \ \ j = 1,2,...,n, \ \ j \neq k; \\ m_{kk} = \dfrac{1}{a_{k+1,k}^{(n-k-1)}}. \end{cases} \quad (3.8)$$

$$M_k^{-1}: \begin{cases} m_{ij} = e_{ij}, \ i = 1,2,...,n, \ \ j = 1,2,...,n, \ \ i \neq k; \\ m_{kj} = a_{k+1,j}^{(n-k-1)}, \ \ j = 1,2,...,n. \end{cases} \quad (3.9)$$

Несложно доказать, что у подобных матриц собственные числа совпадают. Далее для матрицы P строится характеристический полином

$$D(\lambda) = \det(P - \lambda E) = (-1)^n[\lambda^n - p_1\lambda^{n-1} - p_2\lambda^{n-2} - \ldots - p_n]. \quad (3.10)$$

Это полином степени n. Очевидно, что он имеет n корней $\lambda_1, \lambda_2, \ldots, \lambda_n$. Некоторые из них могут быть кратными, при этом выполняется соотношение (3.2). Необходимо не только найти все корни полинома, но и определить их кратность (см. п. 3.1.3).

3.1.2 Вычисление собственных векторов методом Данилевского

Далее для каждого собственного числа вычисляется соответствующий ему собственный вектор. Собственные векторы у подобных матриц не совпадают. Если y_i – это собственный вектор матрицы P, соответствующий собственному числу λ_i, то

$$x_i = Sy_i, \ i = 1, 2, \ldots, n. \quad (3.11)$$

При этом собственный вектор матрицы P выглядит следующим образом:

$$y_i = \begin{pmatrix} \lambda_i^{n-1} \\ \lambda_i^{n-2} \\ \dots \\ \lambda_i \\ 1 \end{pmatrix}. \qquad (3.12)$$

3.1.3 Определение кратности собственных чисел и векторов

При поиске кратных корней возникают некоторые сложности. Дело в том, что если кратность корня четная, то в этой точке наблюдается экстремум (минимум или максимум) характеристического полинома, а если нечетная – то полином просто меняет знак. Пример приведен на рис. 3.1.

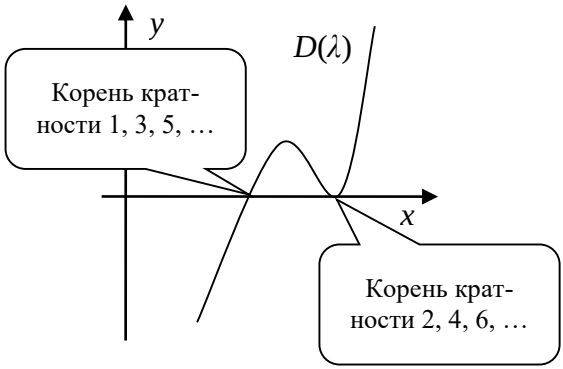

Рисунок 3.1 – Поведение характеристического полинома

Согласно определению [1], корень уравнения ξ имеет кратность k, если не только функция в точке ξ принимает нулевое значение, но и k –1 ее производных:

$$f^{(i)}(\xi) = 0,\ i = 0, 1, 2, \dots, k–1. \qquad (3.13)$$

При $i = 0$ имеем саму функцию. Таким образом, получаем k нулей функции и ее производных.

Учитывая погрешности вычислений на ЭВМ, при четной кратности корня характеристический полином может пройти либо выше, либо ниже нулевой отметки (рис. 3.2).

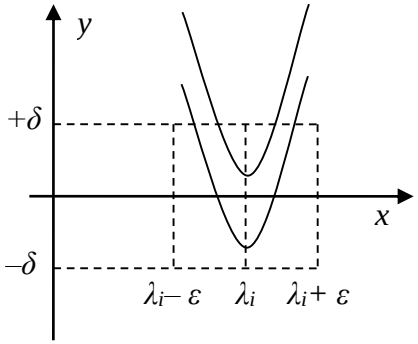

Рисунок 3.2 – Погрешности при вычислении собственных чисел

Здесь ε и δ – достаточно малые числа. Т.о., программа может либо вообще не найти корня, либо найти сразу два. Поэтому договоримся считать корнем любое число λ_i, для которого $|f(\lambda_i)| < \delta$. При этом, если два корня λ_{i1} и λ_{i2} расположены близко друг к другу (т.е. $|\lambda_{i1} - \lambda_{i2}| < 2\varepsilon$), то корнем следует считать только один из них, либо за корень принять число, расположенное между ними:

$$\lambda_i = (\lambda_{i1} + \lambda_{i2})/2. \qquad (3.14)$$

Поиск собственных чисел продолжается до тех пор, пока не будут найдены все, т.е. пока не выполнится условие (3.2).

3.2 Формат входных данных

Формат входного файла:

`m` – тип задачи (1 – поиск собственных чисел, 2 – векторов);

`n` – порядок матрицы;

$a_{11}…a_{1n}$ – коэффициенты матрицы.
$a_{21}…a_{2n}$
……
$a_{n1}…a_{nn}$

3.3 Формат выходных данных

Формат выходного файла:

P	– матрица Фробениуса;
λ_i	– *i*-е собственное число;
$\|A-\lambda_i E\|$	– проверка *i*-го собственного числа (при m = 1);
x_i	– *i*-й собственный вектор (при m = 2);
$Ax_i-\lambda_i x_i$	– проверка *i*-го собственного вектора (при m = 2);
k_i	– кратность *i*-го собственного числа/вектора;
…	и т.д. для всех $i = 1, 2, …, m$.

4 Решение систем нелинейных уравнений

Не всегда системы уравнений, которые приходится решать в различных задачах, бывают линейными. Для решения систем нелинейных уравнений (СНУ) существует ряд специальных методов для их решения. По аналогии с решением уравнений с одной переменной, можно заключить, что численные методы позволяют быстрее получить приближенное решение при помощи ЭВМ. А также СНУ большой размерности аналитически очень тяжело решаются (если аналитическое решение вообще существует, что, как было показано выше, наблюдается далеко не всегда).

В матричном виде СНУ выглядит следующим образом:

$$f(x) = 0, \tag{4.1}$$

где $f = (f_1, f_2, \ldots, f_n)^T$, $x = (x_1, x_2, \ldots, x_m)^T$, т.е.

$$\begin{cases} f_1(x_1, x_2, ..., x_m) = 0 \\ f_2(x_1, x_2, ..., x_m) = 0 \\ \quad ... \\ f_n(x_1, x_2, ..., x_m) = 0 \end{cases}$$

Если $n < m$, то система может иметь множество решений. Если $n > m$, то система переопределена. В этом случае у нее может не быть решений. Мы будем рассматривать ситуацию с $n = m$. В этом случае количество решений зависит от вида системы функций F. Какое именно решение будет найдено, зависит от начальной точки x^0.

Очевидно, что при $n = m = 1$ получим обычное уравнение с одной переменной. В принципе, все рассмотренные методы в таком случае вырождаются в методы решения

уравнений с одной переменной (с двумя из них мы уже ознакомились ранее). Аналогией производной при $n > 1$ выступает матрица Якоби

$$W(x) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \dots & \dots & \dots & \dots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}. \qquad (4.2)$$

При $n = 1$ якобиан вырождается в обычную производную.

4.1 Методы решения

Для решения СНУ предлагаются три метода – Ньютона, итераций и наискорейшего спуска.

4.1.1 Метод Ньютона

Итерационный процесс, по аналогии с формулами метода Ньютона для решения уравнений с одной переменной (1.14) и (1.17), выглядит следующим образом:

$$x^{(k+1)} = \Phi(x^{(k)}), \qquad (4.3)$$

где

$$\Phi(x^{(k)}) = x^{(k)} - W^{-1}(x^{(k)})\, f(x^{(k)}). \qquad (4.4)$$

Критерий окончания итерационного процесса, по аналогии с (1.13), выглядит так:

$$\left\| x^{(k+1)} - x^{(k)} \right\| < \varepsilon. \qquad (4.5)$$

4.1.2 Метод итераций

Как и метод Ньютона, метод итераций решения СНУ является обобщением метода итераций решения уравнений с одной переменной и имеет вид (4.3). Анализируя (1.14) и (1.18), можно заключить, что для повышения скорости сходимости матрицу Якоби в (4.4) нужно вычислять не в точке $x^{(k)}$, а в некоторой другой точке. Очевидно, что в данном случае определить ее гораздо труднее. Поэтому обычно просто берут точку $x^{(0)}$:

$$\Phi(x^{(k)}) = x^{(k)} - W^{-1}(x^{(0)})f(x^{(k)}). \qquad (4.6)$$

В итоге получаем модифицированный метод Ньютона, и скорость сходимости только падает. Критерий останова определяется выражением (4.5).

4.1.3 Метод наискорейшего спуска

Итерационный процесс строится по общей формуле (4.3), где

$$\Phi(x^{(k)}) = x^{(k)} - \lambda_k \nabla U(x^{(k)}). \qquad (4.7)$$

Функция $U(x)$ преобразует систему функций f в скалярную функцию векторного аргумента:

$$U(x) = f^{\mathrm{T}} \cdot f = \sum_{i=1}^{n} f_i^2(x). \qquad (4.8)$$

Очевидно, что

$$\nabla U(x) = 2(f')^{\mathrm{T}} \cdot f = 2W^{\mathrm{T}}(x) \cdot f(x). \qquad (4.9)$$

Т.е. единственной проблемой остается поиск параметра λ_k. Он должен минимизировать функцию $\Phi(x)$ вдоль направления $\nabla U(x)$:

$$\lambda_k : \quad G(\lambda) = U\left(x^{(k+1)}\right) =$$

$$= U\left(x^{(k)} - \lambda \nabla U\left(x^{(k)}\right)\right) \xrightarrow[\lambda]{} \min. \qquad (4.10)$$

Очевидно, что он должен быть положительным, иначе мы будем двигаться в направлении градиента, а не антиградиента функции (т.е. искать максимум).

Как известно, в точке минимума (как и в других точках экстремума) значение производной функции равно нулю. Используем этот факт для минимизации выражения (4.10):

$$\lambda_k : \ \frac{\partial}{\partial \lambda} U\left(x^{(k)} - \lambda \nabla U\left(x^{(k)}\right)\right) = 0. \qquad (4.11)$$

Уравнение (4.11) можно решить численно, если использовать правила дифференцирования. Можно его решить и аналитически, если прибегнуть к некоторым приближениям. Тогда получим

$$\lambda_k = \frac{1}{2} \frac{g_k^{\mathrm{T}} g_k}{g_k^{\mathrm{T}} W_k^{\mathrm{T}} W_k g_k}, \qquad (4.12)$$

где $g_k = W^{\mathrm{T}}\left(x^{(\mathrm{k})}\right) f\left(x^{(k)}\right)$, $W_k = W\left(x^{(\mathrm{k})}\right)$.

4.2 Формат входных данных

Формат входного файла:

`m` – метод (в порядке их перечисления);
`n` – размерность СНУ;
x^0 – начальное приближение;
`ε` – требуемая погрешность решения;
f_1 – система функций.
f_2
…
f_n

4.3 Формат выходных данных

x^0 – последовательные приближения решения
x^1 СНУ;
…
x^k
ε^* – вектор невязки $f(x^k)$;
$||\varepsilon^*||$ – норма вектора невязки.

5 Интерполирование и численное дифференцирование функций

Приближение функций – одна из наиболее востребованных областей численных методов. Под приближением понимается замена на интервале [a, b] исходной функции $f(x)$ некоторой другой функцией $P(x)$, близкой (по некоторому критерию) к исходной функции. В общем случае, $P(x)$ является полиномом вида

$$P(x) = \sum_{i=0}^{n} c_i \varphi_i(x), \qquad (5.1)$$

где c_i – некоторые действительные константы, а $\varphi_i(x)$ – система действительных линейно-независимых функций. Т.е. любая функция этой системы не может быть представлена в виде линейной комбинации других. Например,

$$\varphi_i(x) = \sin^{i}(x).$$

Задача состоит в том, чтобы, выбрав систему функций, найти такие коэффициенты c_i, при которых отклонение полинома $P(x)$ от исходной функции удовлетворяло бы выдвигаемым критериям. Исходными данными являются узлы x_i, принадлежащие отрезку [a, b] и значения функции в этих узлах $y_i = f(x_i)$, $i = 0, 1, \ldots, m$. При этом полином $P(x)$ называют приближающим или **аппроксимирующим** (от англ. approximate – приблизительный):

$$f(x) = P(x) + R(x), \qquad (5.2)$$

где $R(x)$ – т.н. *остаточный член*.

В данной лабораторной работе мы будем рассматривать такие полиномы, у которых $m = n$.

Например, аппроксимирующий полином можно построить, воспользовавшись методом наименьших квадратов

(МНК). При этом $\varphi_i(x)$ может быть системой любых линейно-независимых функций, а коэффициенты c_i ищутся из условия минимального СКО полученного полинома от исходной функции:

$$\frac{1}{n}\sqrt{\sum_{i=0}^{n}\left(y_i - P\left(x_i\right)\right)^2} \xrightarrow[c_i]{} \min. \qquad (5.3)$$

Картина при этом получается примерно такая, как на рис. 5.1.

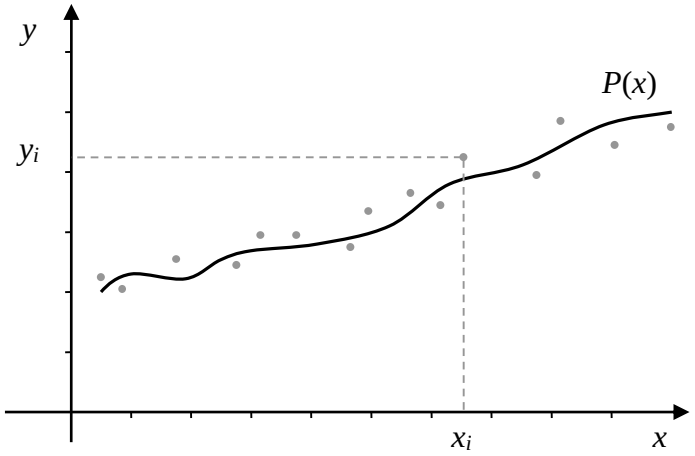

Рисунок 5.1 – Аппроксимация МНК

Если требуется построить такой полином, чтобы он проходил через все точки (x_i, y_i), то его называют **интерполирующим** (от англ. interpolate). Здесь приставка «inter-» имеет смысл «между». Т.е. нас интересует поведение полинома только между точками (x_i, y_i), т.е. между границами отрезка [a, b]. А критерий близости интерполирующего полинома к исходной функции выглядит как

$$y_i = P(x_i). \qquad (5.4)$$

При этом обычно $x_0 = a$, $x_n = b$. Для того же набора точек, что и на рисунке выше, получим результат, показанный на рис. 5.2.

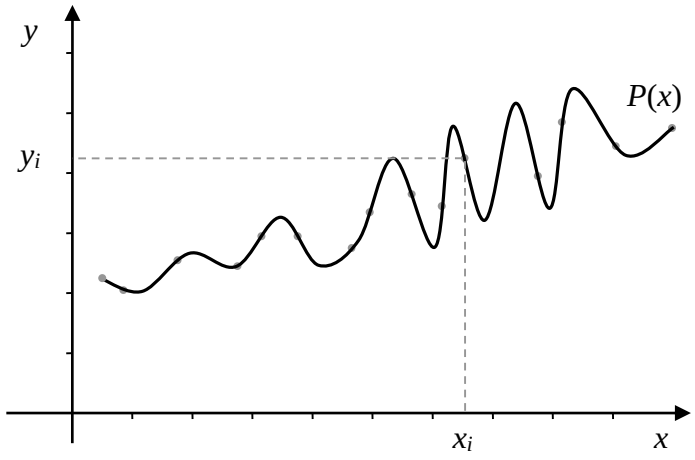

Рисунок 5.2 – Интерполяция методом Ньютона или Лагранжа

На рисунке изображены полиномы Ньютона и Лагранжа (в сущности, это разные формы записи одного и того же полинома степени n), которые мы будем изучать в ходе данной лабораторной работы. Как видно, их недостатком является осцилляция при большом количестве точек. Поэтому их область применения лучше ограничивать теми случаями, когда точек немного. В противном случае нужно пользоваться другими интерполирующими и аппроксимирующими полиномами.

Если же нас интересуют значения полинома $P(x)$ за пределами отрезка $[a, b]$, то такой полином называется **экстраполирующим** (от англ. extrapolate, где приставка «extra» имеет смысл «сверх», «за пределами»).

Аппроксимация функций необходима в двух случаях.

Во-первых, если исходная функция неизвестна. Т.е. имеется только некоторая сетка $\{x_i\}$ и значения функции в узлах сетки $\{y_i\}$. В этом случае говорят, что функция задана *таблично*. Такая ситуация может складываться в любом эксперименте – известно значение искомой характеристики y_i только в некоторых точках x_i в пространстве ее аргументов R^Z, но необходимо иметь возможность найти значения этой характеристики во всех точках некоторого подпространства $X^Z \subset R^Z$. Например, зная давления в некоторых точках трубы с газом, можно выдать прогноз давления по всей трубе. Это поможет найти области падения давления (т.е. нарушения герметичности трубы) или, наоборот, области повышенного давления (что может привести к прорыву трубы в будущем) и оперативно отреагировать на внештатную ситуацию. Или, зная несколько координат некоторого космического тела, движущегося в пространстве, можно построить достаточно гладкий интерполирующий полином, который ответит на вопрос, как выглядела траектория тела в те моменты, когда мы тела не наблюдали (например, оно было закрыто другими космическими телами или находилось за горизонтом, т.е. было невидимо из-за вращения Земли). Если использовать экстраполирующий полином, то можно узнать, как вела себя траектория тела до начала наблюдений, и как она будет вести себя в будущем.

Во-вторых, даже если аналитический вид функции известен, она может иметь очень сложный вид. Существуют различные задачи в физике, математике и пр. науках, где вычисление некоторых функций в одной точке пространства аргументов может занимать от нескольких секунд до

часов, дней и т.д. В этом случае, если время ограничено, вычисляют значение функции только в нескольких узлах (получая табличную функцию) и проводят аппроксимацию или интерполяцию.

Сетка $\{x_i\}$ при $i = 0, 1, \ldots, n$ имеет $n+1$ узел. Она может быть равномерной или неравномерной. Если сетка равномерная (т.е. расстояние между ее соседними узлами одинаковое), то все узлы задавать не обязательно. Достаточно знать начальный узел x_0 и шаг сетки h:

$$x_i = x_0 + ih,\ i = 0, 1, \ldots, n. \qquad (5.5)$$

Если заданы только границы отрезка (точки a и b, или x_0 и x_n), то из (5.5) следует, что $x_n = x_0 + nh$, т.е. шаг можно найти по формуле

$$h = \frac{x_n - x_0}{n} = \frac{b-a}{n}. \qquad (5.6)$$

Все вышесказанное можно отнести также и к задачам численного дифференцирования (заметьте, что, говоря об аппроксимации и упомянутых ее разновидностях, мы не употребляем слово «численная», т.к. это в принципе чисто численные методы). Только в этом случае нас интересует не сама функция, а некоторая ее производная. Поэтому будем заменять производную функции (см. 5.2) производной аппроксимирующего полинома:

$$f^{(k)}(x) = (P(x) + R(x))^{(k)} = P^{(k)}(x) + R^{(k)}(x). \qquad (5.7)$$

В данной лабораторной работе мы будем находить первую и вторую производные полинома $P(x)$. При этом

$$P^{(k)}(x) = \left(\sum_{i=0}^{n} c_i \varphi_i(x) \right)^{(k)} = \sum_{i=0}^{n} c_i \varphi_i^{(k)}(x). \qquad (5.8)$$

5.1 Методы решения

Данная лабораторная работа выполняется по вариантам. Первый вариант – это метод Ньютона, второй – Лагранжа. Эти полиномы являются степенными.

Известно, что через две точки можно провести одну и только одну прямую, через три – одну и только одну параболу и т.д. Поэтому, через n+1 точку $\{x_i\}$ можно провести одну и только одну кривую порядка n. Отсюда можно сделать два вывода. **Во-первых**, чем больше количество точек в заданной сетке, тем выше, в общем случае, будет степень полинома $P(x)$. Именно этим и объясняется осциллирующее поведение полиномов Ньютона и Лагранжа при большом количестве точек – просто их вид становится слишком сложным. Отметим, что для других интерполирующих полиномов это может быть и не так. Например, МНК, независимо от количества точек, дает полином, для которого выполняется условие (5.3). Т.е., если в качестве линейно-независимых функций взять $\varphi_i(x) = x^i$, i = 1, 2, …, m, то можно построить, например, кубический полином для любого количества точек (при m = 3). Порядок у него ниже, поэтому он более гладкий. При $m = n$ МНК становится обычным интерполяционным полиномом. **Во-вторых**, полиномы Ньютона и Лагранжа совпадают, т.е. это просто две формы записи одного и того же полинома, и их можно преобразовать к следующему виду:

$$P_n(x) = \sum_{i=0}^{n} k_i x^i,$$

где k_i – некоторые константы. Индекс n у полинома указывает на его порядок.

При этом, каждым вариантом необходимо реализовать 6 задач:

1) вычисление полинома на равномерной сетке;

2) вычисление полинома на неравномерной сетке;

3) вычисление первой производной полинома на равномерной сетке;

4) вычисление первой производной полинома на неравномерной сетке;

5) вычисление второй производной полинома на равномерной сетке;

6) вычисление второй производной полинома на неравномерной сетке.

При использовании равномерной сетки вводится новая переменная

$$q = \frac{x - x_0}{h}, \tag{5.9}$$

и подставляется в полином и его производные. Таким образом, получается, что они зависят только от q, а x и $\{x_i\}$ явным образом в них не входят. Т.е. имеем $P(q)$. Получить его можно самостоятельно, сделав замену (5.9) в полиноме $P(x)$.

Выигрыш состоит в том, что не нужно хранить в памяти узлы сетки $\{x_i\}$, поэтому ее используется примерно в два раза меньше.

5.1.1 Полином Ньютона

Полином Ньютона имеет вид (5.1), где

$$c_i = [x_0, ..., x_i], \quad \varphi_i(x) = \prod_{j=0}^{i-1} (x - x_j),$$

$$\Rightarrow P_n(x) = \sum_{i=0}^{n} \left([x_0, ..., x_i] \cdot \prod_{j=0}^{i-1} (x - x_j) \right). \tag{5.10}$$

Здесь $\left[x_i,...,x_j\right]=\dfrac{\left[x_{i+1},...,x_j\right]-\left[x_i,...,x_{j-1}\right]}{x_j-x_i}$ – так называемые *разделенные разности*, а $[x_i] = y_i$ (условно).

Запишем первую производную полинома Ньютона, согласно (5.8):

$$P_n'(x)=\sum_{i=0}^{n}\left(\left[x_0,...,x_i\right]\cdot\left(\prod_{j=0}^{i-1}\left(x-x_j\right)\right)'\right)=$$

$$=\sum_{i=1}^{n}\left(\left[x_0,...,x_i\right]\cdot\sum_{j=0}^{i-1}\prod_{\substack{k=0\\k\neq j}}^{i-1}\left(x-x_k\right)\right). \qquad (5.11)$$

Как видно, первое слагаемое, представляющее собой константу, обратилось в 0. Аналогично, во второй производной пропадают первые два слагаемых, т.к. второе слагаемое представляет собой линейную функцию. Из (5.8) имеем:

$$P_n''(x)=\sum_{i=0}^{n}\left(\left[x_0,...,x_i\right]\cdot\left(\prod_{j=0}^{i-1}\left(x-x_j\right)\right)''\right)=$$

$$=\sum_{i=2}^{n}\left(\left[x_0,...,x_i\right]\cdot\sum_{j=0}^{i-1}\sum_{\substack{k=0\\k\neq j}}^{i-1}\prod_{\substack{l=0\\l\neq k\\l\neq j}}^{i-1}\left(x-x_l\right)\right). \qquad (5.12)$$

Вид полинома Ньютона для равномерной сетки предлагается найти самостоятельно. При этом производится замена (5.9), вследствие чего переменная x и сетка $\{x_i\}$ из полинома убираются. Разделенные разности заменяются *конечными разностями*:

$$\Delta^i y_j = \Delta^{i-1} y_{j+1} - \Delta^{i-1} y_j,\ \Delta^0 y_j = y_j. \qquad (5.13)$$

5.1.2 Полином Лагранжа

Полином Лагранжа также имеет вид (5.1), где

$$c_i = \frac{y_i}{\prod\limits_{\substack{j=0 \\ j\neq i}}^{n}\left(x_i - x_j\right)},\ \varphi_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n}\left(x - x_j\right),$$

$$\Rightarrow L_n(x) = \sum_{i=0}^{n} y_i \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}. \qquad (5.14)$$

Запишем первую и вторую производную полинома Лагранжа, согласно (5.8):

$$L_n'(x) = \sum_{i=0}^{n}\left(\frac{y_i}{\prod\limits_{\substack{j=0 \\ j\neq i}}^{n}\left(x_i - x_j\right)}\left(\prod_{\substack{j=0 \\ j\neq i}}^{n}\left(x - x_j\right)\right)'\right) =$$

$$= \sum_{i=0}^{n}\left(\frac{y_i}{\prod\limits_{\substack{j=0 \\ j\neq i}}^{n}\left(x_i - x_j\right)}\sum_{\substack{j=0 \\ j\neq i}}^{n}\prod_{\substack{k=0 \\ k\neq i \\ k\neq j}}^{n}\left(x - x_k\right)\right). \qquad (5.15)$$

$$L_n''(x) = \sum_{i=0}^{n}\left(\frac{y_i}{\prod\limits_{\substack{j=0 \\ j\neq i}}^{n}\left(x_i - x_j\right)}\left(\prod_{\substack{j=0 \\ j\neq i}}^{n}\left(x - x_j\right)\right)''\right) =$$

$$= \sum_{i=0}^{n} \left(\frac{y_i}{\prod\limits_{\substack{j=0 \\ j \neq i}}^{n} \left(x_i - x_j \right)} \sum_{\substack{j=0 \\ j \neq i}}^{n} \sum_{\substack{k=0 \\ k \neq i \\ k \neq j}}^{n} \prod_{\substack{l=0 \\ l \neq i \\ l \neq j \\ l \neq k}}^{n} \left(x - x_l \right) \right). \qquad (5.16)$$

Вид полинома Лагранжа для равномерной сетки также предлагается найти самостоятельно. При этом производится замена (5.9), вследствие чего переменная x и сетка $\{x_i\}$ из полинома убираются.

5.2 Формат входных данных

Формат входного файла:

`k` – порядок производной (0 – вычисляется сам полином, 1 – его первая производная, 2 – вторая производная);

`n` – порядок полинома;

`s` – любой символ или строка, задающая тип исходной сетки (равномерная/неравномерная);

`a b` – границы отрезка (при равномерной сетке);

$x_0 \ldots x_n$ – узлы сетки (если она неравномерная);

$y_0 \ldots y_n$ – значения функции в узлах сетки;

`m` – количество интервалов в результирующей сетке (т.е. количество узлов – $m + 1$, что сделано для унификации с узлами исходной сетки);

$x_0 \ldots x_m$ – узлы результирующей сетки;

`t` – любой символ или строка, сообщающая, известно или нет аналитическое выражение для функции $f(x)$;

`f(x)` – аналитическое выражение для функции (если оно известно).

5.3 Формат выходных данных

Формат выходного файла:

x_0 $P^{(k)}(x_0)$ – значение полинома или его производных в узлах результирующей сетки;
x_1 $P^{(k)}(x_1)$
...
x_m $P^{(k)}(x_m)$

ε – СКО (если аналитическое выражение для функции известно).

6 Приближение сплайнами

Приближение сплайнами – еще один способ построения интерполирующих полиномов. В отличие от полиномов Ньютона и Лагранжа, степень которых зависит от количества узлов в исходной сетке, при построении сплайна его степень может варьироваться.

Так, мы можем построить линейные, параболические и кубические сплайны для сеток с произвольным количеством узлов. Следовательно, мы избавляемся от одного из недостатков интерполирующих полиномов, рассмотренных выше – сплайны имеют несложный математический вид и не осциллируют на сетках с большим количеством узлов (рис. 6.1).

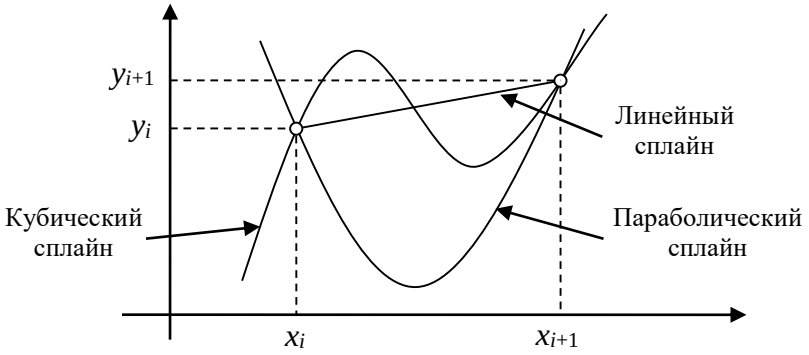

Рисунок 6.1 – Приближение сплайнами

Итак, сплайн строится между двумя узлами сетки. Если он линейный, то это прямая линия, если параболический – парабола, если кубический – кривая третьего порядка. Т.е. от количества узлов зависит только количество сплайнов, но не их порядок. Таким образом, для сетки $\{x_i\}$

из n+1 узла (i = 0, 1, …, n) имеем n сплайнов S_i(x), i = 1, 2, …, n–1, аргумент x должен лежать в интервале от x_i до x_{i+1}.

Как мы уже знаем, по двум точкам прямая линия строится однозначно. Чтобы построить параболу, нужно либо задать еще одну точку, либо ввести так называемое *граничное условие*. Это значение не самой функции, а некоторой ее производной в одной из границ отрезка. В параболических сплайнах применяется первая производная. Чтобы построить кубическую кривую, надо либо задать еще две точки, либо ввести два граничных условия. В кубических сплайнах задают значение либо первой, либо второй производной в обеих границах отрезка. Хотя возможны и другие комбинации – значение первой и второй производной в одной из границ отрезка и т.п.

Очевидно, что для вычисления интерполированного значения в некоторой точке x необходимо определить, в область определения какого именно сплайна это значение попадает. Из рис. 6.1 видно, что за пределами своей области определения значения сплайнов перестают интерполировать функцию с достаточной точностью.

Для решения задач численного дифференцирования точности сплайнов уже не хватает.

6.1 Методы решения

Для выполнения лабораторной работы необходимо реализовать в программе интерполяцию табличной функции линейными, кубическими и параболическими сплайнами.

Сплайны, как и все интерполирующие полиномы, имеют вид (5.1), но количество узлов n не равно порядку

сплайна m. Однако, следует учесть, что количество самих сплайнов равно n–1, т.е.

$$S_i(x) = \sum_{j=0}^{m} c_{ji}\varphi_{ji}(x),\ \ \varphi_{ji}(x) = (x - x_i)^j$$

$$\Rightarrow S_i(x) = \sum_{j=0}^{m} c_{ji}(x - x_i)^j,\ \ i = 0,1,...,n-1,\ \ m = 1,2,3. \quad (6.1)$$

Чтобы не использовать константы c с двумя индексами, вводят *коэффициенты сплайна* $a_i = c_{0i}$, $b_i = c_{1i}$, $c_i = c_{2i}$, $d_i = c_{3i}$.

6.1.1 Линейные сплайны

Согласно (6.1), линейный сплайн имеет вид

$$S_i(x) = a_i + b_i(x - x_i),\ \ i = 0,1,...,n-1. \quad (6.2)$$

При этом

$$a_i = y_i,\ \ b_i = \frac{\Delta y_i}{\Delta x_i}. \quad (6.3)$$

Здесь $\Delta x_i = x_{i+1} - x_i$, $\Delta y_i = y_{i+1} - y_i$.

6.1.2 Параболические сплайны

Из (6.1) получаем выражение для параболического сплайна:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2,\quad i = 0,1,...,n-1. \quad (6.4)$$

Для коэффициентов a_i и c_i имеем

$$a_i = y_i,\ \ c_i = \frac{b_{i+1} - b_i}{2\Delta x_i}. \quad (6.5)$$

Граничным условием для параболического сплайна является значение первой производной функции в первой либо последней точке отрезка, т.е. $A_i = f'(x_i)$, где $i = 0$ или n.

Во входном файле задается точка (0 или n) и значение производной в ней.

Если задано число A_0, то коэффициенты b_i ищутся, начиная с b_0:

$$b_0 = A_0,\ b_{i+1} = \frac{2\Delta y_i}{\Delta x_i} - b_i,\ i = 0,1,...,n-2. \qquad (6.6)$$

Если задано число A_n, то коэффициенты b_i ищутся, начиная с b_n:

$$b_n = A_n,\ b_i = \frac{2\Delta y_i}{\Delta x_i} - b_{i+1},\ i = n-1, n-2,...,0. \qquad (6.7)$$

6.1.3 Кубические сплайны

По аналогии, из (6.1) получаем для кубического сплайна

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3,\quad i = 0,1,...,n-1. \qquad (6.8)$$

Коэффициенты ищутся следующим образом:

$$a_i = y_i,\ c_i = \frac{1}{2}M_i,\ d_i = \frac{M_{i+1} - M_i}{6\Delta x_i}. \qquad (6.9)$$

Для коэффициентов b_i

$$b_i = \frac{\Delta y_i}{\Delta x_i} - \frac{\Delta x_i}{6}(2M_i + M_{i+1}),\ i = 0,1,...,n-1 \qquad (6.10)$$

или

$$b_i = \frac{\Delta y_{i-1}}{\Delta x_{i-1}} - \frac{\Delta x_{i-1}}{6}(2M_i + M_{i-1}),\ i = 1,2,...,n. \qquad (6.11)$$

Неизвестные M_i находятся из решения СЛАУ

$$AM = g, \qquad (6.12)$$

где

$$A = \begin{cases} a_{jj} = \dfrac{1}{3}\left(\Delta x_{j-1} + \Delta x_j\right), \;\; j = 1, 2, \ldots, n-1; \\ a_{j,\,j+1} = a_{j+1,\,j} = \dfrac{1}{6}\Delta x_j, \;\; j = 1, 2, \ldots, n-2; \end{cases} \tag{6.13}$$

$$g_i = \frac{\Delta y_i}{\Delta x_i} - \frac{\Delta y_{i-1}}{\Delta x_{i-1}} - \beta_1 - \beta_{n-1}, \;\; i = 1, 2, \ldots, n-1,$$

$$\text{где} \;\; \beta_1 = \begin{cases} \dfrac{\Delta x_0}{6} B_0, \;\; i = 1 \\ 0, \; i \neq 1 \end{cases} \tag{6.14}$$

$$\beta_{n-1} = \begin{cases} \dfrac{\Delta x_{n-1}}{6} B_n, \;\; i = n-1 \\ 0, \; i \neq n-1 \end{cases}$$

$$M = (M_1, M_2, \ldots, M_{n-1}).$$

Для кубического сплайна можно выбрать любой тип граничных условий (либо по первой, либо по второй производной). Соответственно, во входном файле будут находиться значения первой (A_0 и A_n) или второй (B_0 и B_n) производной в первой и последней точке отрезка.

Если граничные условия заданы по второй производной, то $M_0 = B_0$, $M_n = B_n$, а остальные неизвестные M_i находятся решением СЛАУ (6.12).

Если граничные условия заданы по первой производной, то $b_0 = A_0$, $b_n = A_n$. Тогда к системе можно добавить еще два уравнения, используя (6.10) при $i = 0$ и (6.11) при $i = n$, а также перенести в левую часть СЛАУ слагаемые с неизвестными коэффициентами из выражений для g_1 и g_{n-1}. Получим модифицированную СЛАУ

$$\tilde{A}M = \tilde{g}, \tag{6.15}$$

где

$$\tilde{A} = \begin{cases} \tilde{a}_{00} = \dfrac{\Delta x_0}{3};\ \tilde{a}_{nn} = \dfrac{\Delta x_{n-1}}{3}; \\ \tilde{a}_{jj} = \dfrac{1}{3}\left(\Delta x_{j-1} + \Delta x_j\right),\ \ j = 1,2,...,n-1; \\ \tilde{a}_{j,j+1} = \tilde{a}_{j+1,j} = \dfrac{1}{6}\Delta x_j,\ \ j = 0,1,...,n-1; \end{cases} \qquad (6.16)$$

$$\tilde{g} = \begin{cases} \tilde{g}_0 = \dfrac{\Delta y_0}{\Delta x_0} - A_0;\ \tilde{g}_n = A_n - \dfrac{\Delta y_{n-1}}{\Delta x_{n-1}}; \\ \tilde{g}_i = \dfrac{\Delta y_i}{\Delta x_i} - \dfrac{\Delta y_{i-1}}{\Delta x_{i-1}},\ \ i = 1,2,...,n-1; \end{cases} \qquad (6.17)$$

$$M = (M_0, M_1, M_2, \ldots, M_n).$$

Трехдиагональные СЛАУ (6.12) и (6.15) можно решать любым методом решения СЛАУ. Однако, учитывая их структуру, оптимальным будет использование *метода прогонки*.

6.1.4 Метод прогонки

Пусть имеется трехдиагональная СЛАУ $Ax = b$ размера $n \times n$. Ее решение методом прогонки строится следующим образом:

$$x_i = \frac{1}{v_i}\left(r_i - a_{i,i-1}x_{i-1}\right),\ \ i = 1,2,...,n; \qquad (6.18)$$

$$r_i = b_i - \frac{a_{i,i+1}}{v_{i+1}}r_{i+1},\ \ v_i = a_{ii} - \frac{a_{i,i+1}a_{i+1,i}}{v_{i+1}}, \quad i = n-1, n-2, ..., 1. \qquad (6.19)$$

При этом полагаем, что

$$v_n = a_{nn},\ \ r_n = b_n,\ \ a_{10} = 0,\ \ x_0 = 0. \qquad (6.20)$$

6.2 Формат входных данных

Формат входного файла:

k	– порядок сплайна (1 – линейный, 2 – параболический, 3 – кубический);
n	– количество сплайнов;
$x_0 \ldots x_n$	– узлы сетки;
$y_0 \ldots y_n$	– значения функции в узлах сетки;
i A_i	– граничные условия (для $k = 2$);
B_0 B_n	– граничные условия (для $k = 3$);
m	– количество интервалов в результирующей сетке (т.е. количество узлов – $m + 1$, что сделано для унификации с узлами исходной сетки);
$x_0 \ldots x_m$	– узлы результирующей сетки;
t	– любой символ или строка, сообщающая, известно или нет аналитическое выражение для функции $f(x)$;
f(x)	– аналитическое выражение для функции (если оно известно).

6.3 Формат выходных данных

Формат выходного файла:

a_0 b_0 c_0 d_0 a_1 b_1 c_1 d_1 ... a_{n-1} b_{n-1} c_{n-1} d_{n-1}	– коэффициенты сплайнов (естественно, что коэффициенты c указываются только для $k = 2$ и $k = 3$, коэффициенты d – только для $k = 3$);
x_0 S(x_0) x_1 S(x_1) ...	– значение сплайна в узлах результирующей сетки;

x_m $S(x_m)$

ε – СКО (если аналитическое выражение для функции известно).

7 Численное интегрирование функций

Численное интегрирование функций – весьма важный раздел численных методов. При помощи интегралов решается широкий спектр практических задач, самые распространенные из которых – вычисление объемов и площадей тел, длин кривых и т.д. Помимо очевидного преимущества ЭВМ при проведении сложных расчетов, вспомним еще тот факт, что не все интегралы имеют первообразную, а значит, не все интегралы могут быть вычислены аналитически.

В данной лабораторной работе мы будем находить интегралы двумя способами. Первый заключается в интегрировании интерполяционных полиномов. Т.е. исходная функция заменяется некоторым интерполяционным полиномом, который легко интегрировать:

$$I = \int_{\alpha}^{\beta} f(x)\,dx \approx \int_{\alpha}^{\beta} \sum_{i=0}^{k} c_i \varphi_i(x)\,dx = \sum_{i=0}^{k} c_i \int_{\alpha}^{\beta} \varphi_i(x)\,dx =$$

$$= \sum_{i=0}^{k} c_i \, \Phi_i(x)\Big|_{\alpha}^{\beta} = \sum_{i=0}^{k} I_i. \qquad (7.1)$$

По аналогии с интерполяционными полиномами, для этого класса методов численного интегрирования задается исходная сетка $\{x_i\}$ и значение функции в узлах сетки $\{y_i\}$, $i = 0, 1, \ldots, n$. Если сетка равномерная, то достаточно знать границы отрезка a и b, а узлы при необходимости вычисляются по формулам (5.5) и (5.6).

Второй способ заключается нахождении интеграла на отрезке [–1, 1] с подбором оптимальных узлов интегрирования:

$$I' = \int_{-1}^{1} f(t)\,dt = \sum_{i=1}^{n} c_i f(t_i). \qquad (7.2)$$

Узлы t_i подбираются таким образом, чтобы формула (7.2) была точной для степенного полинома максимально возможного порядка. При переходе к отрезку [a, b] имеем

$$I = \int_a^b f(x)\,dx = \frac{b-a}{2}\sum_{i=1}^{n} c_i f(x_i), \qquad (7.3)$$

$$x_i = \frac{b+a}{2} + \frac{b-a}{2} t_i. \qquad (7.4)$$

Существуют и другие подходы к вычислению интегралов. Например, статистические, или вероятностные (как и вероятностные методы решения СЛАУ, различные модификации этих методов называются методами Монте-Карло). Например, вычислить объем шара радиуса R статистически можно следующим образом. Будем случайным образом задавать N точек (x_i, y_i, z_i), лежащие в кубе, в который вписан шар (т.е. каждая из координат должна лежать в диапазоне [$–R$, R]). Подсчитаем также количество точек M, оказавшихся внутри шара, т.е. для которых выполняется условие

$$x_i^2 + y_i^2 + z_i^2 \le R^2,$$

Очевидно, что отношение объемов куба и шара будет приблизительно пропорционально отношению общего количества точек и количества точек, попавших внутрь шара:

$$\frac{V_K}{V_{Ш}} \approx \frac{N}{M}.$$

Чем больше количество точек N, тем точнее будет выполняться данное соотношение, т.е.

$$\lim_{N\to\infty} \frac{N}{M} = \frac{V_K}{V_{Ш}}.$$

Учитывая, что $V_K = 8R^3$, получим

$$V_{Ш} = 8R^3 \cdot \frac{M}{N}.$$

7.1 Методы решения

Предлагается реализовать четыре обязательных метода численного интегрирования функций – левосторонних и правосторонних прямоугольников, трапеций, Симпсона и, по желанию, один из двух дополнительных – Чебышева или Гаусса.

7.1.1 Формулы прямоугольников

В *формуле левосторонних прямоугольников* полагаем, что на отрезке $[x_i, x_{i+1}]$ функция $\varphi_i(x) = 1$, $c_i = y_i$ (рис. 7.1).

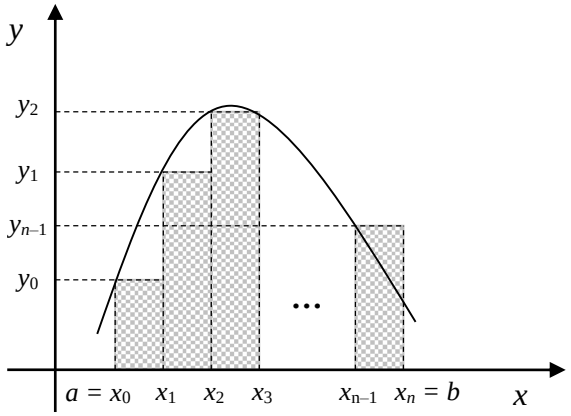

Рисунок 7.1 – Интегрирование левосторонней формулой прямоугольников

Очевидно, что $\Phi_i(x) = x$, $k = n–1$. Тогда из (7.1) получаем:

$$I = \int_{x_0}^{x_n} f(x)\,dx \approx \sum_{i=0}^{n-1} y_i\, x\Big|_{x_i}^{x_{i+1}} = \sum_{i=0}^{n-1} y_i h_i, \qquad (7.5)$$

$$h_i = \Delta x_i = x_{i+1} - x_i.$$

Если сетка равномерная, то

$$I = \int_{x_0}^{x_n} f(x)\,dx \approx h\sum_{i=0}^{n-1} y_i. \qquad (7.6)$$

В *формуле правосторонних прямоугольников* полагаем, что на отрезке $[x_i, x_{i+1}]$ функция $\varphi_i(x) = 1$, $c_i = y_{i+1}$ (рис. 7.2).

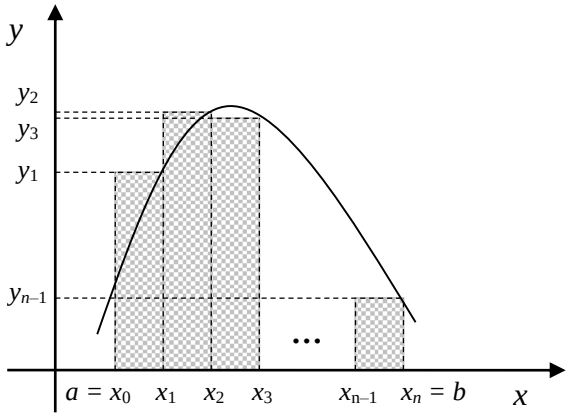

Рисунок 7.2 – Интегрирование правосторонней формулой прямоугольников

Тогда $\Phi_i(x) = x$, $k = n–1$, и из (7.1) получаем

$$I = \int_{x_0}^{x_n} f(x)\,dx \approx \sum_{i=0}^{n-1} y_{i+1}\, x\Big|_{x_i}^{x_{i+1}} = \sum_{i=0}^{n-1} y_{i+1} h_i. \qquad (7.7)$$

Если сетка равномерная, то

$$I = \int_{x_0}^{x_n} f(x)\,dx \approx h\sum_{i=0}^{n-1} y_{i+1} = h\sum_{i=1}^{n} y_i. \qquad (7.8)$$

7.1.2 Формула трапеций

В *формуле трапеций* полагаем, что функция на отрезке $[x_i, x_{i+1}]$ заменяется прямой линией, соединяющей точки (x_i, y_i) и (x_{i+1}, y_{i+1}) (рис. 7.3).

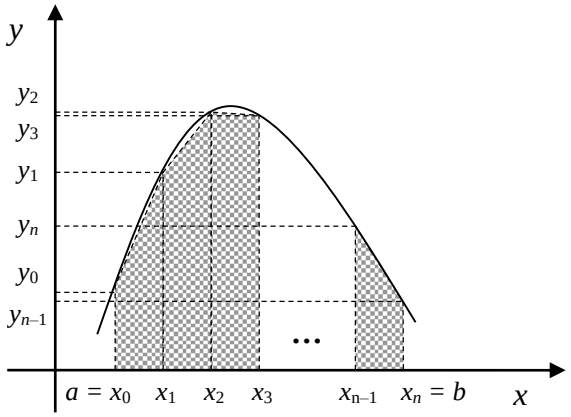

Рисунок 7.3 – Интегрирование формулой трапеций

Несложно записать уравнение прямой, проходящей через две точки:

$$c_i\varphi_i(x) = \frac{y_{i+1} - y_i}{h_i}(x - x_i) + y_i.$$

Интегрируем:

$$c_i\Phi_i(x) = \frac{y_{i+1} - y_i}{2h_i}(x - x_i)^2 + y_i x$$

$$\Rightarrow\ I_i = c_i\Phi_i(x)\Big|_{x_i}^{x_{i+1}} = \frac{1}{2}(y_{i+1} + y_i)h_i. \qquad (7.9)$$

Это же выражение можно легко получить из геометрических соображений (см. рис. 7.3).

Есть и еще один способ вывода данной формулы. Очевидно, что на каждом интервале функция заменяется поли-

номом первого порядка. Нам уже известны полиномы, интерполирующие табличную функцию по p+1 точке и дающие при этом степенной полином порядка p – это полиномы Ньютона и Лагранжа. Как уже было сказано, они являются разной формой записи одного и того же полинома, поэтому их применение даст одинаковый результат. Возьмем, например, полином Лагранжа. Тогда

$$I_r = \int_{x_r}^{x_{r+p}} L_p(x)\,dx = \int_{x_r}^{x_{r+p}} \left(\sum_{i=r}^{r+p} y_i \prod_{\substack{j=r \\ j\neq i}}^{r+p} \frac{x - x_j}{x_i - x_j} \right) dx =$$

$$= \sum_{i=r}^{r+p} y_i \int_{x_r}^{x_{r+p}} \prod_{\substack{j=r \\ j\neq i}}^{r+p} \frac{x - x_j}{x_i - x_j}\,dx = \sum_{i=r}^{r+p} y_i A_i = \sum_{i=0}^{p} y_{r+i} A^*_{r+i}. \quad (7.10)$$

Здесь A^* – некоторые *квадратурные коэффициенты*. Если сетка равномерная, то делаем замену (5.9):

$$I_r = \int_{x_r}^{x_{r+p}} L_p(q)\,dx = h \int_{r}^{r+p} L_p(q)\,dq = h \sum_{i=0}^{p} y_{r+i} A_i. \quad (7.11)$$

Т.к. сетка равномерная, квадратурные коэффициенты не зависят от индекса r. Используем выражение (5.6) и введем новые коэффициенты H_i:

$$I_r = \frac{(x_{r+p} - x_r)}{p} \sum_{i=0}^{p} y_{r+i} A_i = ph \sum_{i=0}^{p} y_{r+i} H_i, \quad (7.12)$$

$$\text{где } H_i = \frac{1}{p} \int_0^p L_p(q)\,dq. \quad (7.13)$$

Коэффициенты H_i называются *коэффициентами Ньютона-Котеса*. Для построения полинома первого порядка нужны всего две точки (т.е. p = 1), поэтому сетку можно считать равномерной. Интегрируя (7.13), получим

$$H_0 = H_1 = \frac{1}{2} \;\Rightarrow\; I_r = h_r\left(\frac{1}{2}y_r + \frac{1}{2}y_{r+1}\right). \qquad (7.14)$$

Т.е. полученное выражение совпадает с (7.9). Остается только просуммировать по всем интервалам:

$$I = \sum_{i=0}^{n-1} I_i = \frac{1}{2}\sum_{i=0}^{n-1} h_i\left(y_i + y_{i+1}\right). \qquad (7.15)$$

Если сетка равномерная, то

$$I = \frac{h}{2}\sum_{i=0}^{n-1}\left(y_i + y_{i+1}\right) = \frac{h}{2}\left(y_0 + 2\sum_{i=1}^{n-1} y_i + y_n\right). \qquad (7.16)$$

7.1.3 Формула Симпсона

Для повышения точности интегрирования можно использовать полиномы более высокого порядка. Так, для $p = 2$ получаем *формулу Симпсона*. Полином (парабола) строится по трем точкам, поэтому имеет значение, равномерная сетка или нет. Отрезки для интегрирования берутся парами, поэтому количество интервалов интегрирования должно быть четным (т.е. $n = 2m$, $m = 1, 2, \ldots$). Интегрируя (7.10), получаем

$$I_r = \frac{h_{r+1} + h_r}{6h_{r+1}h_r}\left(h_{r+1}\left(2h_r - h_{r+1}\right)y_r + \right.$$

$$\left. + \left(h_{r+1} + h_r\right)^2 y_{r+1} + h_r\left(2h_{r+1} - h_r\right)y_{r+2}\right),$$

$$I = \sum_{r=0}^{m-1} I_{2r}. \qquad (7.17)$$

Для равномерной сетки по формулам (7.12) и (7.13) получаем (при этом все $h_r = h$):

$$H_0 = \frac{1}{6},\; H_1 = \frac{4}{6},\; H_2 = \frac{1}{6}$$

$$\Rightarrow I_r = 2h\left(\frac{1}{6}y_r + \frac{4}{6}y_{r+1} + \frac{1}{6}y_{r+2}\right). \qquad (7.18)$$

Суммируем по всем интервалам:

$$I = \frac{2h}{6}\sum_{i=0}^{m-1}\left(y_{2i} + 4y_{2i+1} + y_{2i+2}\right) =$$

$$= \frac{2h}{6}\left(y_0 + 4\sum_{i=0}^{m-1}y_{2i+1} + 2\sum_{i=1}^{m-1}y_{2i} + y_{2m}\right). \qquad (7.19)$$

7.1.4 Формула Чебышева

Формула Чебышева получается из несколько других соображений (7.2-7.4). При этом

$$c_i = \frac{2}{n} \;\Rightarrow\; I = \frac{b-a}{n}\sum_{i=1}^{n} f\left(x_i\right). \qquad (7.20)$$

Узлы x_i находятся согласно (7.4). При этом абсциссы точек интегрирования t_i находятся как решение СНУ:

$$t_1^i + t_2^i + \ldots + t_n^i = \frac{n\left[1+\left(-1\right)^i\right]}{2\left(i+1\right)},\; i = 1,2,\ldots,n. \qquad (7.21)$$

Формула Чебышева является точной для всех полиномов до степени n включительно. Недостатком формулы Чебышева является то, что система (7.21) не имеет действительных решений при $n = 8$ и при $n \geq 10$.

7.1.5 Формула Гаусса

Формула Гаусса также соответствует выражениям (7.2–7.4). Выглядит она идентично формуле (7.3):

$$I = \frac{b-a}{2}\sum_{i=1}^{n} A_i f\left(x_i\right). \qquad (7.22)$$

Узлы x_i, аналогично, находятся согласно (7.4), а абсциссы точек интегрирования t_i являются нулями *полинома Лежандра*

$$P_n(t) = \frac{1}{2^n \cdot n!} \frac{d^n}{dt^n} \left(t^2 - 1\right)^n . \qquad (7.23)$$

Известно, что эти нули действительны, различны и лежат на отрезке [–1, 1]. Коэффициенты A_i определяются решением СЛАУ

$$\sum_{i=1}^{n} A_i t_i^{j-1} = \frac{1-(-1)^j}{j}, \quad j = 1, 2, \ldots, n. \qquad (7.24)$$

Определитель этой системы есть определитель Вандермонда

$$D = \prod_{i=1}^{n} \prod_{j=i+1}^{n} \left(t_j - t_i\right) \neq 0,$$

следовательно, система (7.24) имеет единственное решение.

Формула Гаусса является точной для всех полиномов до степени $2n–1$ включительно.

7.1.6 Вычисления с заданной точностью

Рассмотрим случай, когда необходимо вычислить интеграл с заданной точностью, при этом точное значение интеграла не известно. В этом случае сначала интеграл считается на некоторой начальной сетке с количеством интервалов интегрирования $n_0 = n$. Обозначим полученное значение интеграла как I_0. Затем, аналогично, на сетке с количеством интервалов $n_1 = \alpha \cdot n_0$ ($\alpha > 1$) находим значение интеграла I_1. Считая, что значение I_1 найдено с большей точностью (т.к. сетка более частая), условно примем его за точное значение. Тогда относительную погрешность интегрирования можно оценить по формуле

$$\delta = \left| \frac{I_1 - I_0}{I_1} \right|. \qquad (7.25)$$

Если она удовлетворяет заданной погрешности, то вычисления можно прекращать, иначе добавляем в сетку новые узлы и продолжаем процесс. В общем случае,

$$n_k = \alpha \cdot n_{k-1} = \alpha^k \cdot n_0, \qquad (7.26)$$

а процесс завершается при

$$\delta = \left| \frac{I_k - I_{k-1}}{I_k} \right| < \varepsilon, \quad k = 1, 2, \ldots \qquad (7.27)$$

Для упрощения разбиения отрезка интегрирования на интервалы, часто полагают $\alpha = 2$.

Примечания.

1. Формула для расчета относительной погрешности даст деление на ноль, если какой-либо из интегралов I_k получится равным нулю. Тогда погрешность интегрирования можно оценить по формуле для абсолютной погрешности:

$$\Delta = \left| I_k - I_{k-1} \right|. \qquad (7.28)$$

2. Т.к. в методах Чебышева и Гаусса количество отрезков интегрирования влияет на размер системы уравнений для поиска коэффициентов, то порядок формулы фиксируется, а повышение точности интегрирования достигается ее кратным применением на заданном отрезке.

7.2 Формат входных данных

Формат входного файла:

`m` – формула интегрирования (в порядке их перечисления в п. 7.1), при $m = 5$ используется дополнительный метод;

`g` – любой символ или строка, задающие тип сетки: равномерная, неравномерная, динамическая (при $m \neq 5$);

`n` – количество интервалов интегрирования (если используется формула Симпсона, то кратно двум);

`k` – порядок формулы Чебышева или Гаусса (при $m = 5$);

`a b` – границы отрезка (если сетка не является неравномерной или $m = 5$);

$x_0 \ldots x_n$ – узлы сетки (если она неравномерная);

`s` – любой символ или строка, определяющие способ задания функции, если сетка не динамическая и $m \neq 5$ (табличная, аналитическая);

$y_0 \ldots y_n$ – значения функции в узлах сетки (если она задана таблично);

`f(x)` – аналитическое выражение для функции (если сетка динамическая или $m = 5$);

ε – точность вычисления интеграла на динамической сетке.

7.3 Формат выходных данных

Формат выходного файла:

`I` – значение интеграла;

`k` – количество итераций (для динамической сетки);

ε^* – достигнутая точность (для динамической сетки);

`t_i` – абсциссы точек интегрирования (при m = 5);

`A_i` – коэффициенты A_i для формулы Гаусса.

8 Решение обыкновенных дифференциальных уравнений

О необходимости численных методов решения уравнений и систем уравнений мы уже говорили. Рассмотрим ситуацию, когда уравнения и системы уравнений включают дифференциалы. Отметим также, что не все ДУ имеют аналитическое решение, например,

$$y' = x^2 + y^2.$$

Другой пример. Уравнение

$$y' = \frac{y - x}{y + x}$$

имеет решение

$$\frac{1}{2}\ln\left(x^2 + y^2\right) + \operatorname{arctg}\frac{y}{x} = C.$$

Здесь (и далее) C – произвольная константа. Т.о., хотя ДУ и имеет решение, но выразить в чистом виде функцию $y(x)$ из него невозможно.

В общем случае, ОДУ имеет следующий вид:

$$y^{(n)} = f\left(x, y(x), y'(x), ..., y^{(n-1)}(x)\right). \qquad (8.1)$$

Его решением является семейство функций $y(x) + C$. Фиксируем одну из них, удовлетворяющую n начальным условиям

$$\begin{cases} y(x_0) = y_0, \\ y'(x_0) = y_0', \\ ..., \\ y^{(n-1)}(x_0) = y_0^{(n-1)}. \end{cases} \qquad (8.2)$$

В дальнейшем для сокращения формул вместо $y^{(i)}(x)$ будем использовать запись $y^{(i)}$.

Если речь идет о системе ОДУ, то имеем

$$y_k^{(n)} = f_k\left(x, y_1, ..., y_p, y_1', ..., y_p', y_1^{(n-1)}, ..., y_p^{(n-1)}\right), \quad k = 1, 2, ..., p. \tag{8.3}$$

Ее решением является семейство функций $y_k(x) + C_k$. Фиксируем систему из p функций, удовлетворяющих $p \cdot n$ начальным условиям

$$\begin{cases} y_k\left(x_0\right) = y_{0k}, \\ y_k'\left(x_0\right) = y_{0k}', \\ ..., \\ y_k^{(n-1)}\left(x_0\right) = y_{0k}^{(n-1)}. \end{cases} \tag{8.4}$$

8.1 Методы решения

В данной лабораторной работе будем применять *методы Рунге-Кутта* для решения ОДУ первого порядка, решения систем ОДУ и решения ОДУ n-го порядка.

8.1.1 Решение ОДУ первого порядка

ОДУ первого порядка, согласно (8.1), имеет вид

$$y' = f(x, y).$$

Т.е. просто полагаем $n = 1$. При этом задано начальное условие $y_0 = y(x_0)$.

Решение ОДУ первого порядка методом Рунге-Кутта выглядит следующим образом:

$$y\left(x_{i+1}\right) = y\left(x_i\right) + \sum_{j=1}^{q} p_j k_j\left(h_i\right), \tag{8.5}$$

где q – порядок точности. Будем рассматривать 4 порядка точности. При этом

$$k_j\left(h_i\right)=h_i\cdot f\left(x_i+\alpha_j h_i, y\left(x_i\right)+\sum_{r=1}^{j-1}\beta_{jr}k_r\left(h_i\right)\right). \qquad (8.6)$$

При q = 1 (первый порядок точности) имеем

$$\begin{cases} p_1=1, \\ \alpha_1=0. \end{cases} \qquad (8.7)$$

При q = 2 (второй порядок точности) коэффициент p_1 можно выбрать любой в диапазоне $[0,1)$, а далее

$$\begin{cases} p_2=1-p_1, \\ \alpha_1=0,\ \alpha_2=\beta_{21}=\dfrac{1}{2p_2}. \end{cases} \qquad (8.8)$$

Например,

$$\begin{cases} p_1=\dfrac{1}{2},\ p_2=\dfrac{1}{2}, \\ \alpha_1=1,\ \beta_{21}=1 \end{cases}$$

или

$$\begin{cases} p_1=0,\ p_2=1, \\ \alpha_1=\dfrac{1}{2},\ \beta_{21}=\dfrac{1}{2}. \end{cases}$$

При q = 3 (третий порядок точности)

$$\begin{cases} p_1=p_3=\dfrac{1}{6},\ p_2=\dfrac{4}{6}, \\ \alpha_1=0,\ \alpha_2=\dfrac{1}{2},\ \alpha_3=1, \\ \beta_{21}=\dfrac{1}{2},\ \beta_{31}=-1,\ \beta_{32}=2. \end{cases} \qquad (8.9)$$

При q = 4 (четвертый порядок точности)

$$\begin{cases} p_1 = p_4 = \dfrac{1}{6},\ \ p_2 = p_3 = \dfrac{2}{6}, \\ \alpha_1 = 0,\ \ \alpha_2 = \dfrac{1}{2},\ \ \alpha_3 = \dfrac{1}{2},\ \ \alpha_4 = 1, \\ \beta_{21} = \dfrac{1}{2},\ \ \beta_{31} = 0,\ \ \beta_{32} = \dfrac{1}{2}, \\ \beta_{41} = 0,\ \ \beta_{42} = 0,\ \ \beta_{43} = 1. \end{cases} \tag{8.10}$$

8.1.2 Решение систем ОДУ

Пусть имеется система ОДУ (8.3) и начальные условия (8.4). Поскольку в общем случае решение получается громоздким, будем рассматривать систему ДУ первого порядка (далее – СДУ), т.е. $n = 1$:

$$y'_k = f_k\left(x, y_1, y_2, ..., y_p\right),\ \ y_{0k} = y_k\left(x_0\right), \quad k = 1, 2, ..., p. \tag{8.11}$$

По аналогии с (8.5), решение СДУ будет иметь вид

$$y_k\left(x_{i+1}\right) = y_k\left(x_i\right) + \sum_{j=1}^{q} p_j k_{jk}\left(h_i\right), \tag{8.12}$$

$$k_{jk}\left(h_i\right) = h_i \cdot f_k\left(x_i + \alpha_j h_i, y_1\left(x_i\right) + \sum_{r=1}^{j-1} \beta_{jr} k_{r1}\left(h_i\right), \right.$$

$$\left. y_2\left(x_i\right) + \sum_{r=1}^{j-1} \beta_{jr} k_{r2}\left(h_i\right), ..., y_p\left(x_i\right) + \sum_{r=1}^{j-1} \beta_{jr} k_{rp}\left(h_i\right) \right). \tag{8.13}$$

Коэффициенты p, α и β ищутся по формулам (8.7–8.10) для соответствующего порядка точности.

8.1.3 Решение ОДУ n-го порядка

Имеем ОДУ n-го порядка (8.1) с граничными условиями (8.2). Введем обозначения

$$\begin{cases} y(x) = y_1(x), \\ y'(x) = y_2(x), \\ ..., \\ y^{(n-1)}(x) = y_n(x). \end{cases} \tag{8.14}$$

Очевидно, что

$$\begin{cases} y_i'(x) = y_{i+1}(x), \ i = 1, 2, ..., n-1; \\ y_n'(x) = f(x, y_1, y_2, ..., y_n). \end{cases} \tag{8.15}$$

Таким образом, мы получили СДУ (8.11), в которой

$$\begin{cases} f_i(x, y_1, y_2, ..., y_n) = y_{i+1}, \ i = 1, 2, ..., n-1; \\ f_n(x, y_1, y_2, ..., y_n) = f(x, y_1, y_2, ..., y_n). \end{cases} \tag{8.16}$$

Полученную систему решаем согласно (8.12) и (8.13).

8.2 Формат входных данных

Формат входного файла:

`t` – тип задачи (в том порядке, в котором они рассмотрены в п. 8.1);

`p` – количество уравнений в СДУ (при $t = 2$);

`n` – порядок ДУ (при $t = 3$);

`q` – порядок точности;

`g` – любой символ или строка, задающие тип сетки (равномерная, неравномерная);

`m` – количество интервалов;

`a b` – границы отрезка (если сетка равномерная);

$x_0 \ldots x_m$ – узлы сетки (если она неравномерная);

y_0 – граничные условия (количество определяется типом задачи);

`f` – аналитическое выражение для функции (8.1) при $t = 1$ или $t = 3$;

f_1, f_2, …, f_p	– система функций (8.3) при $t = 2$.
a	– любой символ или строка, сообщающая, известно или нет точное аналитическое решение $y(x)$ или $y_k(x)$;
y	– точное аналитическое решение $y(x)$ или $y_k(x)$ (если оно известно).

Для того, чтобы воспользоваться модулем, вычисляющим значение аналитической функции, все переменные задачи нужно свести к векторному аргументу x: $x_1 = x$, $x_2 = y$, $x_3 = y'$ и т.д.

8.3 Формат выходных данных

Формат выходного файла:

x_0 y_0 x_1 y_1 … x_m y_m	– значения искомой функции в узлах сетки (при $t = 1$ или $t = 3$);
x_0 y_{10}…y_{p0} x_1 y_{11}…y_{p1} … x_m y_{1m}…y_{pm}	– значения искомых функций в узлах сетки (при $t = 2$);
ε	– СКО (если известно аналитическое решение).

9 Решение линейных интегральных уравнений

Опять же, нет необходимости обосновывать очевидную потребность в численных методах решения уравнений. В данной лабораторной работе будем рассматривать уравнения, содержащие интегралы. Ограничимся случаем, когда неизвестная функция входит в интеграл линейно, т.е. классом линейных интегральных уравнений (ЛИУ).

Уравнение вида

$$\int_a^b K(x,s)\cdot y(s)\,ds = f(x) \qquad (9.1)$$

называется ЛИУ Фредгольма 1-го рода. Здесь $f(x)$ – правая часть, x принадлежит некоторому интервалу $[c, d]$; $y(s)$ – искомая функция, s принадлежит некоторому интервалу $[a, b]$; $K(x, s)$ – ядро уравнения, заданное на прямоугольнике $[a \le s \le b, c \le x \le d]$.

Уравнение вида

$$y(x) - \lambda\int_a^b K(x,s)\cdot y(s)\,ds = f(x) \qquad (9.2)$$

называют ЛИУ Фредгольма 2-го рода. Здесь λ – некоторая константа, а x и s заданы на одинаковом интервале $[a, b]$. Соответственно, ядро задано на квадрате $[a \le s \le b, a \le x \le b]$.

9.1 Методы решения

Будем решать ЛИУ Фредгольма 1-го и 2-го рода, применяя в каждом случае методы последовательных приближений и дискретизации.

9.1.1 Метод последовательных приближений

Предположим, что решение ЛИУ Фредгольма 2-го рода (9.2) можно представить в виде

$$y(x)=\sum_{i=0}^{\infty}\lambda^i\varphi_i(x), \tag{9.3}$$

$$\begin{cases}\varphi_0=f(x),\\ \varphi_i=\int\limits_a^b K(x,s)\cdot\varphi_{i-1}(s)\,ds,\ \ i=1,2,...\end{cases} \tag{9.4}$$

Если

$$|\lambda|<\frac{1}{M(b-a)},\ \ M=\max_{a\le x,s\le b}|K(x,s)|, \tag{9.5}$$

то ряд (9.3) сходится.

Т.к. мы не можем численно вычислить сумму бесконечного ряда, ограничимся m его членами:

$$y(x)\approx y_m(x)=\sum_{i=0}^{m}\lambda^i\varphi_i(x). \tag{9.6}$$

Параметр m подбирается таким образом, чтобы погрешность формулы (9.6) не превышала заранее заданной величины ε. Погрешность формулы (9.6) определяется выражением

$$\varepsilon^*=N\frac{\left(M(b-a)\cdot|\lambda|\right)^m}{1-M(b-a)\cdot|\lambda|},\ \ N=\max_{a\le x\le b}|f(x)|. \tag{9.7}$$

9.1.2 Метод дискретизации

Введем сетку по переменным x и s:

$$\begin{cases}a=x_0<x_1<...<x_n=b\\ a=s_0<s_1<...<s_n=b\end{cases}\ \Rightarrow$$

$$\int_a^b K(x_i,s)\cdot y(s)\,ds = \sum_{j=0}^{n} A_j K_{ij} y_j,$$

$$K_{ij} = K(x_i, s_j),\ \ y_j = y(s_j). \tag{9.8}$$

Здесь A_j – квадратурные коэффициенты. Тогда вместо (9.2) получим СЛАУ

$$\sum_{j=0}^{n}\left(e_{ij} - \lambda A_j K_{ij}\right) y_j = f_i,\ \ f_i = f(x_i), \tag{9.9}$$

или, в матричном виде,

$$By = f,\ \ B = E - \lambda KA. \tag{9.10}$$

9.1.3 Решение ЛИУ первого рода

В общем случае, ЛИУ Фредгольма 1-го рода можно свести ко 2-му роду, тогда вместо (9.1) получим

$$\begin{cases} \alpha\cdot y(x) + \int_a^b \tilde{K}(x,s)\cdot y(s)\,ds = \tilde{f}(x), \\ \tilde{K}(x,s) = \int_c^d K(t,x)\cdot K(t,s)\,dt, \\ \tilde{f}(x) = \int_c^d K(t,x)\cdot f(t)\,dt. \end{cases} \tag{9.11}$$

Очевидно, что модифицированное ядро задано уже на квадрате $[a \le s \le b,\ a \le x \le b]$, как и ядро уравнения (9.2). Далее задача решается рассмотренными выше методами. Остается единственная проблема – поиск положительного параметра α. Для этого оценим невязку решения ЛИУ:

$$\varepsilon^* = \sqrt{\int_c^d \left[f(x) - \int_a^b K(x,s)\cdot y_\alpha(s)\,ds \right]^2 dx}. \tag{9.12}$$

Если полученная невязка удовлетворяет заданной погрешности, то считаем задачу решенной. Таким образом,

решаем задачу (9.11) при различных значениях α, пока очередное решение $y_\alpha(x)$ не станет достаточно точным.

Для простоты положим $c = a$ и $d = b$.

9.2 Формат входных данных

Формат входного файла:

`q` – тип ЛИУ;

`p` – метод решения (в порядке их перечисления в п. 9.1);

`a b` – отрезок, на котором заданы переменные x и s;

`K(x,s)` – ядро ЛИУ;

`f(x)` – правая часть ЛИУ;

`λ` – параметр ЛИУ (при $q = 2$);

`n` – количество интервалов, на которое разбиваются отрезки;

`ε` – требуемая точность решения (если выбран метод последовательных приближений).

`a` – любой символ или строка, сообщающая, известно или нет точное аналитическое решение $y(x)$;

`y` – точное аналитическое решение $y(x)$ (если оно известно).

9.3 Формат выходных данных

Формат выходного файла:

x_0 y_0 – значения искомой функции в узлах сетки;

x_1 y_1

…

x_n y_n

ε – СКО (если известно аналитическое решение).

Список литературы

1. Мицель, А. А. Вычислительные методы: учебное пособие / А. А. Мицель. – Томск : В-Спектр, 2010. – 264 с.

2. Мицель, А. А. Вычислительные методы: учебное пособие / А. А. Мицель. – Томск : Эль Контент, 2013. – 198 с.

3. Мицель, А. А. Практикум по численным методам / Мицель А. А. – Томск : ТУСУР, 2004. – 196 с.

4. Бахвалов, Н. С. Численные методы / Н. С. Бахвалов, Н. П. Жидков, Г. М. Кобельков. – Москва : БИНОМ. Лаборатория знаний, 2006. – 636 с.

5. Образовательный стандарт вуза ОС ТУСУР 01-2021. Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления [Электронный ресурс]: база нормативных документов ТУСУР. – Томск : ТУСУР, 2021. – 52 с. URL: https://regulations.tusur.ru/documents/70.

6. Положение о проверке самостоятельности выполнения письменных работ бакалавров, специалистов и магистров в ТУСУР [Электронный ресурс]: база нормативных документов ТУСУР. – Томск : ТУСУР, 2016. – 10 с. URL: https://regulations.tusur.ru/documents/81.

Приложение А
Дополнительные материалы

А.1. Порядок выполнения и сдача работы

Выполнение и сдача лабораторных работ состоит из следующих шагов:

1. Изучение теоретического и практического материала по теме работы. Изучение материала происходит во время лекционных занятий, а также при самостоятельном изучении методических материалов [1-3].

2. Разработка и защита программной части лабораторной работы. Защита заключается, во-первых, в демонстрации того, что входные и выходные данные программы соответствуют оговоренному формату и результаты вычислений корректны. Во-вторых, проверяется степень владения исходным кодом программы. Могут быть заданы вопросы по некоторым частям программы, либо будет предложено внести в программу некоторые модификации.

3. Написание и защита отчета по лабораторной работе. Требования к структуре отчета приведены ниже. Оформляется отчет согласно требованиям образовательного стандарта ОС ТУСУР [5]. Защита отчета заключается в демонстрации владения теоретическим материалом (знание и вывод формул, доказательство утверждений и т.п.).

Согласно принятому положению о проверке самостоятельности выполнения письменных работ бакалавров [6], лабораторная работа выполняется студентом самостоятельно, без заимствования чужого кода.

На выполнение каждой работы отводится две недели (на некоторые, более сложные – месяц). Защищать работы

не обязательно в том порядке, в котором они перечислены. Если к моменту защиты уже получены задания, например, на лабораторные работы №1, №2 и №3, то защищать можно любую из них.

Для тестирования программ можно использовать примеры из [2-4]. Для изучения **краткой** теории изданы учебные пособия [1-2]. Краткой – потому что для полного понимания изучаемых методов информации, полученной из учебных пособий, может оказаться недостаточно. Для получения более подробной информации необходимо изучать дополнительные литературные источники по численным методам.

А.2 Входные и выходные данные программ

Для удобства тестирования, все входные данные для программ должны находиться в текстовых файлах. Обычно файл с входными данными имеет имя «input.txt», а файл с выходными данными – «output.txt», но имена можно варьировать. Формат таких файлов для каждой лабораторной работы оговорен в задании (см. п. 2.1–2.9). При разработке программ с графическим интерфейсом или веб-интерфейсом допускается реализовать ввод данных через графические элементы управления.

Основными типом входных данных программ являются числа (целые и вещественные), а также функции, заданные в аналитическом виде (т.е. строки).

А.2.1 Форматирование чисел и строк

Рассмотрим работу с числами. Для чтения чисел из текстового файла используются стандартные процедуры или операторы ввода:

```
read(f, x); { Pascal }
fscanf(f, format, &x); // C
f >> x; // C++
x = type.Parse(f.ReadLine()); // C#
```

При этом файловая переменная `f` должна быть связана с входным файлом, а указатель в файле должен находиться перед считываемым числом. В языке Pascal эта связь создается последовательным вызовом функций `Assign` и `Reset`, а в C – `fopen` (библиотека `stdio.h`). В языке C++ переменная `f` должна являться экземпляром класса `ifstream` (библиотека `fstream.h`) и связывается с файлом либо при вызове конструктора класса, либо при помощи метода `open`. Тип переменной `x` должен соответствовать типу считываемого числа. При этом все разделители (пробелы, табуляции, переносы строки) пропускаются автоматически. В языке C тип числа задается явным способом – при помощи текстового параметра `format`. Для целого числа это обычно «`%d`» (int) или «`%ld`» (long), для вещественного числа – «`%f`» (float) или «`%lf`» (double). Переменная в этом случае передается в функцию по адресу. Как именно формируется адрес – не важно. Например, адрес ячейки с номером `i` в массиве-векторе `a` можно записать как `a + i`, что эквивалентно `&a[i]`(следовательно, `&a[0]` эквивалентно просто `a`).

Несколько отличается подход, принятый в языке C#. В этом языке существует множество способов считать текстовый файл (статические методы `File.ReadAllLines`, `File.ReadAllText`, `File.ReadLines` статического же класса `System.IO.File`, метод `ReadLine` класса `System.IO.StreamReader` и т.д.). Но все данные считыва-

ются только как строки, которые затем необходимо конвертировать в нужный тип либо вызовом метода `Parse` или `TryParse`, использовать класс `System.Convert` и т.д. При этом, если в строке находилось не одно значение, а несколько, предварительно эту строку необходимо разбить на массив строк, используя метод `Split` строки (типа `string` или класса `System.String`).

Для записи чисел в текстовый файл используются стандартные процедуры или операторы вывода:

```
write(f, x); { Pascal }
fprintf(f, format, x); // C
f << x; // C++
f.Write(x); // C#
```

Файловая переменная `f` должна быть связана с выходным файлом. В языке Pascal эта связь создается последовательным вызовом функций `Assign` и `Rewrite`, а в C – также `fopen`. В языке C++ переменная `f` должна являться экземпляром класса `ofstream` (библиотека `fstream.h`) и связывается с файлом либо при вызове конструктора класса, либо при помощи метода `open`. В языке C#, опять же, имеется много способов записи текстовых данных в файл (методы `File.WriteAllLines` и `File.WriteAllText` класса `System.IO.File`, методы `WriteLine` и `Write` класса `System.IO.StreamWriter` и т.д.).

При этом вывод можно форматировать. Так, в языке Pascal для форматирования чисел при выводе используется запись `x:n`, где `n` – количество позиций для вывода числа. Лишние позиции (не занятые цифрами числа) заполняются пробелами. Это удобно использовать при выводе матриц. Для вещественных чисел можно задавать дополнительный

параметр форматирования: x:n:m, где m – количество десятичных знаков после запятой.

Функции семейства printf языка С обладают более гибкими возможностями форматирования вывода. Во-первых, поддерживается множество форматов вещественных чисел:

- с фиксированной точкой (%f);
- с плавающей точкой, или экспоненциальный (%e) – числа выводятся всегда с экспонентой;
- основной (%g) – автоматически подбирает максимально удобный вид числа.

Во-вторых, позволяют задавать ширину поля – %nz, %n.mz (n и m имеют тот же смысл, что и в Pascal, z – требуемый формат). Если использовать запись %0nz или %0n.mz, то неиспользуемые позиции слева заполнятся нулями. Если использовать знак «минус» (%-nz или %-n.mz), то выравнивание происходит не по правой, а по левой границе поля.

В языке С++ существуют процедуры и манипуляторы для форматирования вывода (setw, setprecision, ios::flags, ios::setf, ios::unsetf, ios::width, ios::fill, ios::precision и многие другие). Манипуляторы setw и setprecision определены в библиотеке iomanip.h, а класс ios – в библиотеке iostream.h. Все классы потокового ввода-вывода являются его наследниками. В случае необходимости, можно также записать число в требуемом формате в строку процедурой sprintf (или используя класс std::stringstream) а затем уже полученную строку подать на выход. Для выравнивания значения по левой или правой границе поля используются

манипуляторы `left` и `right`, или соответствующие флаги метода `ios::setf`.

В языке C# форматирование можно осуществлять разными способами:

```
x.ToString("zm");
String.Format("{0,n:zm}", x);
$"{x,n:zm}";
f.Write("{0,n:zm}", x);
f.Write($"{x,n:zm}");
```

и т.д. В первых трех случаях получаем строку, которую потом можно записать в файл или использовать другим образом. В двух последних случаях строка сразу пишется в файл. По аналогии с предыдущими примерами, здесь `z` – какой-то формат (например, `f`, `e` или `g` – их смысл тот же, что и в языке C/C++), `n` – ширина поля, `m` – точность.

Похожим образом обстоит дело и со строками. При чтении строк посредством функции `fscanf` языка C нужно только помнить, что строковая переменная сама по себе является указателем на область памяти, в которой расположены символы строки, поэтому применение операции извлечения адреса не требуется. Поэтому и объявляться строка должна так же, как массив-вектор. Например:

```
char s1[20];
char s2[] = "qwerty";
char *s3;
char *s4 = "error!";
```

В первом случае создается строка на 20 символов, во втором – требуемое количество символов подсчитывается компилятором. В данном случае это 7 (добавляется символ конца строки). Эту запись можно представить в том виде, в котором инициализируются числовые массивы:

```
char s2[] = {'q', 'w', 'e', 'r', 't',
             'y', '\0'};
```

В третьем случае строка не создается, объявляется только указатель. Если мы хотим в эту строку что-либо поместить, то нужно предварительно выделить память, а после того, как надобность в ней отпадает, освободить:

```
s3 = new char [20];
// используем s3
delete [] s3;
```

Квадратные скобки явно указывают компилятору на то, что удаляется именно массив. В четвертом случае память не выделена, поэтому приведенная запись ошибочна. Происходит попытка занести данные по неинициированному указателю.

В других языках такой проблемы нет. Так, в языке Pascal тип `string` является не массивом, а встроенным типом языка. В языке C++ имеется специальный класс для работы со строками – `std::string`. В языке C# также имеется такой класс, его имя – `String` (соответствующее ему ключевое слово языка – `string`).

При чтении строки, содержащей пробелы, важно учитывать следующее. Язык Pascal считает разделителем строк символ перевода строки, а языки C/C++ по умолчанию – символ пробела. Поэтому нужно грамотно использовать функции `read` и `readln` в языке Pascal, чтобы не считать пустую строку (если указатель в файле находился в конце предыдущей строки). А в языках C/C++ использовать ограничители ввода, если строка может содержать пробелы. Например, формат «`%[^\n]s`» позволит прочитать строку, остановившись на символе конца строки (`\n`). В языке C++ для этого существует функция `getline`. В языке C#,

как уже отмечалось выше, используется другой подход, и разбивать строку на подстроки в любом случае необходимо самостоятельно.

При выводе строк действуют практически те же правила форматирования, что и при выводе целых чисел. Т.е. используется запись `write(f, s:n)` в Pascal или формат `%ns` в C для вывода строки в поле фиксированной ширины. Кроме того, в C можно использовать формат `%-ns`, тогда строка выравнивается (по аналогии с выводом числа) по левому, а не правому краю поля. В языке C++, аналогично, используется манипулятор `setw` или метод `ios::width`. В языке C# – указание ширины поля при форматировании строки.

Для более подробной информации изучайте справочные системы используемого языка программирования.

A.2.2 Работа с функциями, заданными в аналитическом виде

Функции, заданные в аналитическом виде, представляют собой текстовые строки, содержащие:

- математические операции (сложение, вычитание, умножение, деление, возведение в степень);
- функции (sin, cos, tg, ctg, exp, ln, lg);
- константы (числовые, pi, e);
- унарный плюс и минус;
- неизвестные переменные (x – если переменная является скаляром и x1, x2, …, xn – если вектором длиной n);
- круглые скобки.

Для вычисления функций предоставляется специальная библиотека PolStr.

Для языка Pascal она включает модули PolStr.tpu и PolUtils.pas. Второй модуль используется в качестве интерфейса, т.к. заголовочных файлов Pascal не имеет. В файле useps.pas (папка Sample) находится пример программы, использующей данный модуль. Для Delphi она включает модули PolStr.dcu и PolUtils.pas. Соответственно, в папке Sample имеется пример проекта UsePS.dpr. Как в Borland Pascal, так и в Borland Delphi, для использования дополнительного модуля достаточно подключить его к главному при помощи директивы `uses`.

Для языков C/C++ имеется несколько вариантов организации данной библиотеки:

1. Заголовочный и объектный файлы (PolStr.h и PolStr.obj). Для их использования в рамках среды программирования Borland C++ необходимо создать новый проект и включить в него модуль с программой, реализующей задание по лабораторной работе и два перечисленных выше файла. Для примера, в указанной директории помещен проект PS.prj, который включает в себя главный модуль UsePS.cpp и файлы PolStr.h и PolStr.obj. Также имеются отдельные версии заголовочного и объектного файла для сред разработки Borland C++ Builder (проект UsePS.bpr с примером находится в папке sample) и различных версий Visual Studio (причем начиная с Visual Studio 2013, в двух форматах – x86 и x64). Кроме того, имеются версии для компиляторов GCC (аналогично, для режимов x86 и x64), объектный файл в этом случае называется PolStr.o. Формат объектного файла у каждого компилятора отличается, поэтому важно использовать правильную версию файла для успешной сборки проекта.

2. Заголовочный файл и статическая библиотека (библиотека со статической линковкой). Для компиляторов Visual Studio, начиная с версии 2013 – это пара файлов PolStr.h и PolStr.lib, для компиляторов GCC – PolStr.h и PolStr.a. Теоретически, формат статической библиотеки для того или иного компилятора является универсальным, но на практике иногда библиотека, скомпилированная в одной версии компилятора, не собирается в проекте с другим компилятором (особенно это касается Visual Studio).

3. Заголовочный файл и динамическая библиотека – это пара файлов PolStr.h и PolStr.dll. Вообще, для динамической библиотеки заголовочный файл не обязателен, ее функции можно вызывать напрямую. Но представленный заголовочный файл делает этот процесс удобнее. Формат файлов dll является универсальным, и должен подходить для любого компилятора, который умеет с ним работать, и любого языка программирования.

В целом, для Visual Studio представлены по 12 вариантов поставки библиотеки для каждой версии, начиная с 2013, которые отличаются по формату (obj, lib или dll), разрядности (x86 или x64) и конфигурации сборки (debug или release). Для GCC вариантов 8 – obj или lib, x86 или x64. Для каждого варианта имеется проект с примером использования библиотеки.

Для языка C# представлен только один вариант – в виде управляемой динамической библиотеки. Пример ее использования, традиционно, находится в папке Sample. Управляемая динамическая библиотека представляет собой сборку .NET, и не совместима с классическими неуправля-

емыми DLL. Т.е. в C# (и других приложениях .NET) классические DLL использовать можно, а вот использование управляемых сборок в программах на языках, не использующих .NET, является нетривиальной задачей.

Указанные файлы находятся на сетевом диске в директории R:\Romanenko\BM, а также в электронном курсе по дисциплине.

А.2.3 Использование стандартных потоков ввода-вывода

Чаще всего, при тестировании консольной программы удобно, когда данные (все или некоторые) вводятся с клавиатуры, а выводятся, соответственно, на консоль. В финальной же версии консольной программы ввод и вывод традиционно осуществляется через файлы. Поэтому необходимо максимально упростить переключение программы из режима консольного ввода-вывода в режим файлового ввода-вывода и обратно.

В языке Pascal это достигается использованием файловых переменных `input` и `output`. Они соответствуют стандартным виртуальным файлам, отвечающим за ввод и вывод. По умолчанию ввод осуществляется с клавиатуры, а вывод – на консоль. Т.е., следующие записи эквивалентны:

```
write(output, ...) ≡ write(...)
 read(input, ...) ≡ read(...)
```

Поэтому весь ввод и вывод в программе можно осуществлять функциями `read/readln` и `write/writeln` без указания файловой переменной. Когда нужно для ввода и вывода использовать файлы, то достаточно связать с требуемыми файлами переменные `input` и `output`. Когда такая

надобность отпадает, то это связывание помещается в комментарий. Либо наоборот – можно в коде использовать функции `read/readln` и `write/writeln` с указанием файловых переменных. Соответственно, когда нужно для ввода и вывода использовать файлы, эти переменные связываются с требуемыми файлами, а если консоль – необходимо этим переменным присвоить `input` и `output` соответственно.

В языке С стандартные файловые переменные `stdin` и `stdout` защищены от изменения. Поэтому, например, вывод в программе можно осуществлять при помощи функции `fprintf` в некоторый файл `f`. Когда необходимо осуществить вывод в файл, то переменная `f` связывается с требуемым файлом. Когда на консоль – присвоить этой файловой переменной значение `stdout`. Аналогично для ввода. Другой вариант – использование функции `freopen`, которая позволяет связать имеющиеся файловые переменные, в том числе `stdin` и `stdout`, с каким-либо файлом на диске.

Достаточно просто это можно проделать и в С++. В некоторых классах ввода-вывода (`istream_withassign`, `ostream_withassign` и `iostream_withassign`) переопределена операция присвоения. Стандартный ввод осуществляется через `cin` (это определенный в библиотеке `iostream.h` экземпляр класса `istream_withassign`), вывод – через `cout` (экземпляр `ostream_withassign`). Поэтому достаточно присвоить переменным `cin` и `cout` экземпляры классов файлового ввода и вывода соответственно (например, `cin = f`, где `f` – экземпляр класса `ifstream`, связанный с входным файлом). Когда необходимо перейти в режим тестирования, достаточно поместить

в комментарий создание экземпляра класса `f` и указанное присвоение.

Похожий подход используется в языке C#. Чтобы методы `Console.Read`, `Console.Write` и т.п. начали работать не с консолью, а файлом на диске, необходимо использовать методы `Console.SetIn` и `Console.SetOut`, которым в качестве параметров передать экземпляры файловых переменных (например, типа `System.IO.StreamReader` и `System.IO.StreamWriter` соответственно). И, наоборот, чтобы через файловую переменную осуществлять ввод с консоли, ей необходимо присвоить ссылку `Console.In` (это аналог `input/stdin`), а для ввода – соответственно, ссылку `Console.Out` (аналог `output/stdout`).

А.2.4 Результаты вычислений. Погрешность

Если говорить о приближенных числах, то ошибочно было бы считать, что, например, 1.00 = 1. Приближенное число 1.00 соответствует точному числу в диапазоне от 0.995 до 1.005, тогда как 1 – от 0.5 до 1.5. При этом первое число указано с точностью в три десятичных знака, а второе – в один знак.

Следовательно, если при решении задачи численным методом задана погрешность, то ответ должен быть записан так, чтобы было видно, что решение в данную погрешность укладывается. К примеру, если для результата указана абсолютная погрешность $\varepsilon = 0.001$, то результат должен быть представлен в виде 2.912, 0.100 и т.д., но не 4 или 0.52. Зная абсолютную погрешность, можно определить количество знаков после запятой для вывода результата:

$$N = -\lg \varepsilon. \qquad (A.1)$$

Предполагается, что $0 < \varepsilon < 1$. Если число N получается нецелым, то оно округляется до большего целого числа. Затем N используется для форматирования результата.

Если задана относительная погрешность δ, то для определения N можно воспользоваться следующей формулой [1]:

$$N = 1 - \lg(a_m \cdot \delta), \qquad \text{(A.2)}$$

где a_m – первая значащая цифра результата.

Далее, все результаты лабораторных работ требуют проверки. Т.е., помимо самого результата, в выходном файле необходимо поместить доказательство того, что результат верный. Обычно доказательством является тот факт, что абсолютная или относительная погрешность решения меньше заданной погрешности, либо что она близка к нулю. Как именно определяется погрешность решения для каждой лабораторной работы, пояснялось выше.

В зависимости от типа результата, погрешность (*невязка*) может являться скаляром, вектором либо матрицей. При выводе погрешности в файл необходимо использовать научный формат (т.е. ±X.XXXXXE±XX, другое название – экспоненциальный), чтобы, по возможности, **избежать** округлений. Иначе вместо числа –1.12E–15 ($-1.12 \cdot 10^{-15}$), представленного в формате с фиксированной точкой, на консоли увидим малоинформативную надпись «–0.000», ничего не говорящую о порядке погрешности.

Если x – это результат вычислений, а y – точный ответ, то невязка вычисляется по формуле

$$\varepsilon = x - y. \qquad \text{(A.3)}$$

Для получения дополнительной информации о погрешности (невязке), представленной в виде матрицы ε^A

размерности $n \times m$ или вектора ε^b размерности n используется *норма* [1]. Чаще всего она определяется так:

$$\left\|\varepsilon^A\right\| = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}\left(\varepsilon_{ij}^A\right)^2},\ \left\|\varepsilon^b\right\| = \sqrt{\sum_{i=1}^{n}\left(\varepsilon_i^b\right)^2}. \qquad \text{(A.4)}$$

Для скалярной величины s понятие нормы является аналогичным понятию модуля, т.е.

$$\|\varepsilon^s\| = |\varepsilon^s|. \qquad \text{(A.5)}$$

В качестве меры отклонения двух величин друг от друга также используется *среднеквадратичное отклонение* (СКО). Для матриц A и B размерности $n \times m$ и векторов a и b размерности n СКО определяется следующим образом:

$$S_{AB} = \frac{1}{nm}\sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}\left(A_{ij} - B_{ij}\right)^2},$$
$$S_{ab} = \frac{1}{n}\sqrt{\sum_{i=1}^{n}\left(a_i - b_i\right)^2}. \qquad \text{(A.6)}$$

Очевидно, что для скалярных величин x и y

$$S_{xy} = \frac{1}{1}\sqrt{\left(x-y\right)^2} = \left|x-y\right|. \qquad \text{(A.7)}$$

Видно, что СКО – это норма невязки, дополнительно нормированная на количество элементов исследуемого объекта, т.е.

$$S_{AB} = \frac{\left\|\varepsilon^{A-B}\right\|}{nm},\ S_{ab} = \frac{\left\|\varepsilon^{a-b}\right\|}{n},\ S_{xy} = \left\|\varepsilon^{x-y}\right\|. \qquad \text{(A.8)}$$

А.3 Требования к структуре отчета

Отчет должен включать следующие элементы:

1. Титульный лист. Образец титульного листа приведен в приложении Б.
2. Оглавление.

3. Задание на лабораторную работу.

4. Краткую теорию. То есть здесь не требуется подробное описание реализуемых методов, достаточно привести и пояснить все формулы, которые использовались при написании программы.

5. Результаты работы. Результаты могут включать описание разработанной программы или ее алгоритма, результаты тестовых запусков и т.п.

6. Заключение.

7. Список использованных источников.

8. Приложение с листингом программы.

Приложение Б

Образец оформления титульного листа отчета

Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ
(ТУСУР)**

Кафедра автоматизированных систем управления (АСУ)

ТЕМА РАБОТЫ

Отчет по лабораторной работе №__ по дисциплине
«Численные методы»

Выполнил: студент гр. _____

___ _____ И. О. Фамилия

«____» _____ 20__ г.

Проверил: _____

_____ ___ И. О. Фамилия

«____» _____ 20__ г.

Томск 20__