Программирование численных методов на языке C

Маренков Е.Д.

1 Решение квадратных и кубических уравнений

Рассмотрим квадратное уравнение с действительными коэффициентами:

$$ax^2 + bx + c = 0. \tag{1}$$

Его решение дается формулой:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

Ограничимся пока поиском действительных решений.

Напишите функцию, находящие корни по формуле (2)*:*

```
void eq2(double a, double b,
         double c, double *x1, double *x2)
```

Эта формула становится непригодной для использования, если $ac \ll b^2$. Вычисление большего корня всегда дает 0. *Убедитесь в этом на примере уравнения с коэффициентами* $a = 1, b = -1, c = 10^{-30}$.

«Правильный» способ состоит в следующем. Сначала находится величина

$$q = -\frac{1}{2}[b + \mathrm{sgn}(b)\sqrt{b^2 - 4ac}] \tag{3}$$

Тогда корни уравнения определяются как:

$$x_1 = \frac{q}{a}, x_2 = \frac{c}{q}. \tag{4}$$

Создайте функцию, находящие корни этим способом:

```
void eq2_correct(double a, double b,
         double c, double *x1, double *x2)
```

Убедитесь, что получается правильный ответ для случая маленьких ac*.*

В случае отрицательного дискриминанта корни уравнения становятся комплексными. Для работы с комплексными числами, начиная со стандарта C99, введен модуль <complex.h>. См. справку man complex. Он определяет тип double complex и вводит функции для работы с комплексными числами[1].

Усовершенствуйте функцию eq2_ correct, чтобы она могла производить вычисления комплексных корней. Сами коэффициенты по прежнему можно считать действительными.

[1]Хотя библиотека complex.h описана в стандарте C11, ее реализация может отличаться. Так, библиотека, поставляемая с компилятором gcc, вводит полноценную поддержку комплексных чисел, включая арифметические операции над ними. В библиотеке, входящей в состав Microsoft Visual Studio, арифметических операций над комплексными числами нет.

Перейдем к решению кубического уравнения

$$x^3 + ax^2 + bx + c = 0 \tag{5}$$

Снова считаем все его коэффициенты действительными. Вычисления производятся следующим образом [1]. Сначала определяются величины

$$Q = \frac{a^2 - 3b}{9}; \quad R = \frac{2a^3 - 9ab + 27c}{54}; \tag{6}$$

Если $R^2 < Q^3$, уравнение имеет три действительных корня, находящихся по формулам:

$$t = \arccos(R/\sqrt{Q^3}); \tag{7}$$

$$x_1 = -2\sqrt{Q}\cos\frac{t}{3} - a/3; \tag{8}$$

$$x_2 = -2\sqrt{Q}\cos\frac{t + 2\pi}{3} - a/3; \tag{9}$$

$$x_3 = -2\sqrt{Q}\cos\frac{t - 2\pi}{3} - a/3. \tag{10}$$

Если же $R^2 \geq Q^3$, то находится величина

$$A = -\mathrm{sgn}(R)(|R| + \sqrt{R^2 - Q^3})^{1/3} \tag{11}$$

Если $A \neq 0$, то вычисляем

$$B = \frac{Q}{A}$$

Если $A = 0$, то полагаем $B = 0$. Уравнение имеет действительный корень:

$$x_1 = A + B - \frac{a}{3} \tag{12}$$

и два комплексно-сопряженных:

$$x_{2,3} = -\frac{1}{2}(A + B) - \frac{a}{3} \pm i\frac{\sqrt{3}}{2}(A - B) \tag{13}$$

Напишите функцию eq3, вычисляющую три корня кубического уравнения по указанным формулам.

```
void eq3(double a, double b, double c,
         double complex *x1, double complex *x2,
         double complex *x3)
```

Составьте тестовые примеры, демонстрирующие нахождение корней в обоих случаях: когда все три корня действительные и когда два из них комплексные.

Указания

- В файле <math.h> определены математические константы, в частности M_PI [2]. Для нахождения знака числа можно использовать функцию copysign[3]. Полное описание можно получить по справке man math

- Чтобы скомпилировать программу с помощью gcc, можно использовать команду

```
gcc 23eq.c --std=c11 -lm -Wall -o 23eq
```

Опция `-Wall` ("Warnings all") показывает все предупреждения компилятора, исправление которых помогает избежать большего количества ошибок. Опция `-lm` необходима для использования стандартной библиотеки math.

[2] Если используется MS Visual Studio, для использования этих констант необходимо включить #define _USE_MATH_DEFINES. Эта директива должна стоять до включения файла math.h

[3] В MS Visual Studio название этой функции начинается со знака подчеркивания, _copysign

2 Вычисление некоторых специальных функций

В качестве примера рассмотрим интегральную показательную функцию $Ei(x)$, определяемую так:

$$Ei(x) = \int_{-\infty}^{x} \frac{e^t\,dt}{t} \equiv -\int_{-x}^{\infty} \frac{e^{-t}\,dt}{t} \tag{14}$$

Рассмотрим только $x > 0$. Способ вычисления отличается для больших и маленьких значений x. Для маленьких x хорошим приближением является ряд

$$Ei(x) = \gamma + \ln x + \frac{x}{1 \cdot 1!} + \frac{x^2}{2 \cdot 2!} + \cdots \tag{15}$$

Здесь γ - постоянная Эйлера.

Для больших x используется приближение:

$$Ei(x) = \frac{e^x}{x}\left(1 + \frac{1!}{x} + \frac{2!}{x^2} + ...\right) \tag{16}$$

Если ϵ — относительная ошибка, то граница по x определяется величиной $|\ln \epsilon|$.

Создайте функцию

```
double Ei(double x)
```

вычисляющую $Ei(x)$ указанным способом. Для тестирования напишите программу, запрашивающую x и выводящую значение интеграла. Значения интеграла можно посмотреть, например, на https://keisan.casio.com/exec/system/1180573423 При этом необходимо иметь в виду следующее. 1) При вычислении следующего слагаемого суммы нужно использовать значение
предыдущего, чтобы сократить число умножений. 2) Критерием остановки для суммы (15) можно считать $\varepsilon S > t$, где S - уже накопленная сумма, а t – величина добавляемого слагаемого. Для формулы (16) лучшим критерием является $t < \varepsilon$. 3) Необходимо ограничить максимальное число итераций, например, величиной MAXIT. 4) При выборе константы ε, а также при выводе на печать результата работы программы, нужно иметь в виду ограничения типа double, описанные в заголовочном файле <float.h> (см. также man float в качестве справки). 5) Константы MAXIT, EPS, а также постоянную Эйлера удобно описать макросами:

```
#define EULER 0.57721566490153286061
#define MAXIT 100
#define EPS 1.e-15
```

Модифицируйте программу, обеспечив вывод в файл "ei.dat" двух колонок: x и $Ei(x)$. Значения x_{min}, x_{max} и количество точек задаются пользователем при выполнении программы. Для записи в файл используются функции fopen, fclose, fprintf из <stdio.h>.

3 Статистика

Требуется определить статистические характеристики набора данных x_j. Среднее определяется очевидным образом:

$$\bar{x} = \frac{1}{N}\sum x_j \tag{17}$$

(Все суммы берутся для $j = 1 \ldots N$, где N — число величин.)

Дисперсия (variance) равна

$$\sigma^2 = \frac{1}{N-1}\sum (x_j - \bar{x})^2 \tag{18}$$

Заметим, что величина σ часто называется "стандартным отклонением".

Коэффициент асимметрии (skewness) равен:

$$s = \frac{1}{N}\sum \left(\frac{x_j - \bar{x}}{\sigma}\right)^3 \tag{19}$$

Если этот коэффициент положителен, то правое «крыло» распределения длиннее левого.

Коэффициент эксцесса (kurtosis)

$$k = \frac{1}{N}\sum \left(\frac{x_j - \bar{x}}{\sigma}\right)^4 - 3 \tag{20}$$

Он положителен, если пик распределения около математического ожидания острый, и отрицателен, если вершина гладкая.

Файл содержит столбец чисел. Написать программу, вычисляющую для этих данных указанные величины. Программа не должна использовать массивы для хранения данных. Постараться также минимизировать количество просмотров столбца в файле и использованных переменных. Для чтения данных можно использовать функцию fscanf. Признак конца файла можно контролировать функцией feof. Однако, если используется fscanf, в этом нет необходимости, так как fscanf сама возвращает EOF при достижении конца файла. Переместить курсор в начало файла можно функцией rewind. Все эти функции определены в заголовочном файле <stdio.h>.

Для тестирования нужен набор случайных чисел с заранее известными характеристиками. Генераторы таких наборов легко найти в Интернете. Можно предложить использовать в качестве тестового примера нормальное распределение с заданным среднем значением и дисперсией, http://pinetools.com/gaussian-random-number-generator.

4 Вычисление интегралов

Простейшими методами вычисления определенных интегралов являются формула трапеций и правило Симпсона:

$$\int_a^b f(x)\,dx = h\left(\frac{1}{2}f_0 + f_1 + ... + f_{n-1} + \frac{1}{2}f_n\right) + O(1/n^2) \tag{21}$$

$$\int_a^b f(x)\,dx = h\Big[\frac{1}{3}f_0 + \frac{4}{3}f_1 + \frac{2}{3}f_2 + \\ + \frac{4}{3}f_3 + ... + \frac{2}{3}f_{n-2} + \frac{4}{3}f_{n-1} + \frac{1}{3}f_n\Big] + O(1/n^4) \tag{22}$$

Здесь n — число интервалов, на которое делится отрезок интегрирования. Формула трапеций получается заменой функции на каждом интервале отрезком прямой. Поэтому она дает точный ответ для полиномов степени не больше (включая) первой. Формула Симпсона основана на замене функции $f(x)$ в трех последовательных точках многочленом второй степени. Однако, как не трудно видеть, она верна для многочленов степени не больше 3, включая 3.

Чередующиеся коэффициенты 4/3 и 2/3 в (22) на самом деле являются следствием не очень удачного выбора аппроксимирующего многочлена. Можно получить формулу той же точности без чередования коэффициентов [1]:

$$\int_a^b f(x)\,dx = h\Big[\frac{3}{8}(f_0+f_n)+\frac{7}{6}(f_1+f_{n-1})+\frac{23}{24}(f_2+f_{n-2})+ \\ +f_3+f_4+...+f_{n-4}+f_{n-3}\Big]+O(1/n^4) \quad (23)$$

Именно эту формулу, а не (22) рекомендуется использовать на практике, если нужен способ с точностью $O(1/n^4)$.

Написать программу, демонстрирующую использование методов (21) *и* (23). Определить функции:

```
double trapz(finteg f, double a, double b, int n)
double simpson(finteg f, double a, double b, int n)
```

Тип finteg определен как указатель на функцию от одного аргумента:

```
typedef double (*finteg)(double);
```

Убедиться в работоспособности этих функций на тестовых примерах.

Эти формулы позволяют очевидным образом вычислить интеграл от функции, заданной таблицей своих значений. Если же нужно вычислять интеграл от обычной функции, то полезнее алгоритм, позволяющий контролировать погрешность интеграла. Простейший вариант основан на формуле трапеций (21). Пусть q_0 и q_1 — два значения интеграла, полученные с меньшим и большим n. Если $|q_0-q_1|<\varepsilon q_0$, то считается, что количество интервалов разбиения достаточно для достижения точности ε. Так как в (21) входит просто сумма значений функции во внутренних точках отрезка, легко реализовать увеличение количества интервалов без пересчета уже имеющихся значений. Пусть q_0 посчитан на n интервалах. Разделим каждый из интервалов пополам и добавим значения функции в новых точках к уже существующим. Таким образом получится новое значение интеграла q_1, учитывающее результат предыдущей итерации. *Реализовать соответствующий алгоритм в функции*

```
double trapz_adapt(finteg f, double a, double b,
        double eps)
```

5 Простейшие операции с матрицами

Реализуйте следующие функции для работы с матрицами и одномерными массивами. Функции должны быть оформлены отдельными модулем. В заголовочный файл помещаются прототипы функций, а в файл .c — их реализации. Заголовочный файл "linal.h"выглядит так:

```
#ifndef LINAL
#define LINAL

#include <stdlib.h>
#include <math.h>
#include <stdio.h>

double** alloc2d(int n1, int n2);
void free2d(double** a, int n1, int n2);
```

```
double vabs(double* a, int n);
double scal_prod(double* a, double* b, int n);
void fill(double** a, int n1, int n2, double x);
/* n1, n2 - dimensions of matrix a; returns c = a*b */
void matrix_prod(double** a, double** b, double** c,
    int n1, int n2);
void show2d(double** a, int n1, int n2);

#endif
```

Функция alloc2d выделяет память под матрицу размерности $n1 \times n2$; функция free2d освобождает выделенную память; vabs находит модуль (длину) вектора; scal_prod — скалярное произведение двух веторов; fill заполняет матрицу значениями x; matrix_prod вычисляет произведение матриц a и b и возвращает результат в матрице c; show2d выводит матрицу на экран.

Реализации функций поместите в файл "linal.c". *Напишите программу, демонстрирующую работу с модулем.* Модуль подключается так:

```
#include "linal.h"
```

Чтобы скомпилировать код, состоящий из нескольких файлов, достаточно перечислить их все во входе gcc:

```
gcc linal.h linal.c test.c
```

Попробуйте также скомпилировать только модуль отдельно, командой:

```
gcc linal.h linal.c -c
```

На выходе получится объектный файл "linal.o". Этот файл можно использовать при сборке основного файла:

```
gcc test.c linal.o
```

Таким образом можно в больших проектах отдельно компилировать только файлы, в которых произошли изменения.

6 Решение систем линейных уравнений

Дополним модуль с линейной алгеброй из предыдущей задачи двумя процедурами решения систем линейных уравнений.

Наиболее известным способом решения системы из n уравнений является метод исключения неизвестных (метод Гаусса). С помощью первого уравнения можно исключить все x_0 из уравнений $i > 0$. С помощью второго уравнения исключаются все x_1 из третьего и последующих уравнений и т.д. В результате получается система с верхнедиагональной матрицей. Она легко решается методом обратной подстановки, последовательно находящим x для $i = n - 1, ...0$.

Запрограммируйте этот способ решения линейных систем. Алгоритм оформите в виде функции. Приведите тестовый пример использования.

Замечание. На практике метод Гаусса в указанном виде никогда не применяют из-за его неустойчивости к ошибкам округления, проявляющейся когда на диагонале матрицы стоят элементы малой величины. Используется модифицированная версия с выбором ведущего элемента (pivoting). Более удобными методами являются методы декомпозиции, например, LU - разложение, которые позволяют решать сразу набор систем с различными правыми частями.

При решении дифференциальных уравнений часто встречаются системы с трехдиагональной матрицей. Для решения этих систем нужно использовать упрощенный вариант метода Гаусса, иногда называемый «прогонка». Пусть система записана в виде:

$$b_0 x_0 + c_0 x_1 = d_0, \tag{24}$$
$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i, i = 1, \ldots, n-2 \tag{25}$$
$$a_{n-1} x_{n-2} + b_{n-1} x_{n-1} = d_{n-1} \tag{26}$$

Таким образом, главная диагональ состоит из n элементов, от b_0 до b_{n-1}. Поддиагональ задается массивом a_i, а наддиагональ — массивом c_i. Все три массива содержат одинаковое число элементов, так что элементы a_0 и c_{n-1}, не участвующие в расчете, приходится считать равными нулю. Правая часть и решение представлены массивами d_i и x_i, $i = 0, \ldots, n-1$,

В начале находятся коэффициенты

$$c'_i = \frac{c_i}{b_i}, i = 0 \tag{27}$$
$$c'_i = \frac{c_i}{b_i - a_i c'_{i-1}}, i = 1, ..., n-2 \tag{28}$$

и

$$d'_0 = \frac{d_0}{b_0},$$
$$d'_i = \frac{d_i - a_i d'_{i-1}}{b_i - a_i c'_{i-1}}, i = 1, ..., n-1$$

Тогда решение системы имеет вид:

$$x_{n-1} = d'_{n-1},$$
$$x_i = d'_i - c'_i x_{i+1}, i = n-2, ..., 0$$

В общем случае алгоритм прогонки неустойчив. Можно показать, что достаточным условием устойчивости является преобладание диагонали матрицы, то есть

$$|b_i| \geq |a_i| + |c_i| \tag{29}$$

Если это условие не выполнено, то необходимо использовать другие алгоритмы, например, метод Гаусса с выбором ведущего элемента.

Создайте функцию, реализующею метод прогонки и приведите пример использования. Передовать в функцию всю матрицу, в которой большинство элементов — нули, совершенно не оправдано. Вместо этого лучше задать три массива для диагональных элементов и элементов побочных диагоналей:

```
void tridiag(double *a, double *b, double *c,
             double *d, double *sol, int n)
```

7 Кубические сплайны

Пусть известны значения y_i функции $f(x)$ в некоторых точках x_i, $i = 0, \ldots, n$. Задача интерполяции состоит в том, чтобы оценить значение этой функции в произвольной точке x, $x_0 < x < x_n$.

Простейшим способом интерполяции является линейная интерполяция, когда функция на каждом из заданных отрезков заменяется прямой:

$$y = Ay_i + By_{i+1} \tag{30}$$

где коэффициенты A и B, как легко видеть, равны

$$A = \frac{x_{i+1} - x}{x_{i+1} - x_i}, \quad B \equiv 1 - A = \frac{x - x_i}{x_{i+1} - x_i} \tag{31}$$

Одним из недостатков линейной интерполяции является то, что первая производная $f'(x)$ терпит разрыв на границе отрезков $[x_i, x_{i+1}]$, а вторая производная обращается в тех же точках в бесконечность. Кубические сплайны представляют собой способ интерполяции, при котором первая производная является гладкой функцией, а вторая — непрерывной.

Обсуждение различных видов сплайнов можно найти в [2]. Кубический сплайн представляет собой многочлен третьей степени:

$$y = Ay_i + By_{i+1} + Cy''_i + Dy''_{i+1} \tag{32}$$

с коэффициентами (31) и

$$C = \frac{1}{6}(A^3 - A)(x_{i+1} - x_i)^2 \quad D = \frac{1}{6}(B^3 - B)(x_{i+1} - x_i)^2 \tag{33}$$

Значения вторых производных y''_i находятся из решения линейной системы уравнений для $i = 1, \ldots, n-1$

$$\begin{aligned} \frac{x_i - x_{i-1}}{6} y''_{i-1} + \frac{x_{i+1} - x_{i-1}}{3} y''_i + \frac{x_{i+1} - x_i}{6} y''_{i+1} = \\ = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}} \end{aligned} \tag{34}$$

Эта система является трехдиагональной и для ее решения можно использовать метод прогонки, изложенный выше. Чтобы система была полной, необходимо предоставить еще два уравнений для $i = 0, n$. Эти уравнения могут быть получены двумя путями:

1. Обе производные y''_0 и y''_n на концах заданного промежутка считаются равными нулю.

2. Значения вторых производных выбираются так, чтобы первые производные имели заданные величины.

Второй способ дает более аккуратный сплайн на границе. Однако, чтобы не усложнять задачу, будем использовать первый способ.

В файле, состоящим из двух колонок, заданы пары (x, y) значений функции $f(x)$. Используя метод интерполяции кубическими сплайнами, построить таблицу с более мелким шагом.

В программе отдельно выделить функцию

```
void spline(double *x, double *y, int size, double *d2y)
```

которая по заданным в массиве y значениям функции в точках массива x находит массив вторых производных d2y в тех же точках, решая систему уравнений (34).

8 Решение нелинейных уравнений

Речь идет о решении уравнения вида $f(x) = 0$. Среди методов решения можно выделить три: деления пополам, Ньютона и метод простых итераций. Нужно создать модуль roots.c, roots.h, реализующий эти методы.

Метод деления пополам состоит в следующем. Пусть известно, что на отрезке $[a, b]$ функция $f(x)$ имеет ровно один корень. Тогда знаки этой функции на концах отрезка разные. Делим отрезок пополам и выбираем ту половину, в которой функция имеет разные знаки. Корень лежит на этом отрезке. Повторяя деление пополам, можно, в принципе, получить значение корня с любой заданной точностью. Точность определяется длиной последнего отрезка.

Реализуйте функцию rtbis, реализующую этот метод. Прототип функции может иметь вид:

```
double rtbis(double (*func)(double ),
        double x1, double x2, double xacc, int* err)
```

Здесь x_1, x_2 — границы отрезка, *xacc* — требуемая точность, а *err* — код ошибки, равный 1, было превышено допустимое количество итераций. Для проверки можно взять уравнение

$$xe^x = 1 \tag{35}$$

Это уравнение имеет корень $x = 0.56714329$. (См. также функцию Ламберта.)

Метод Ньютона (метод касательных) дается итерационной формулой

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{36}$$

Реализуйте функцию rtnewt, реализующую этот метод. Прототип функции может иметь вид:

```
double rtnewt(void (*funcd)(double, double *,
                            double *),
        double x1, double x2, double xacc, int* err)
```

Второй аргумент функции funcd является значением $f(x)$, а третий — значением производной $f'(x)$. Для тестирования также можно использывать уравнение (35). Сравните количество итераций, необходимое для достижения заданной точности этими двумя методами.

Наконец, метод простых итераций пишется для уравнений вида $g(x) = x$. Он имеет вид:

$$x_{i+1} = g(x_i) \tag{37}$$

Условием сходимости является условие $|g'(x)| < 1$. *Создайте функцию rtiter, реализующую этот метод*:

```
double rtiter(double (*func)(double),
        double x1, double xacc, int* err)
```

9 Решение ОДУ (явные методы)

Пусть есть уравнение первого порядка:

$$\frac{dy}{dx} = f(x, y) \tag{38}$$

с начальным условием

$$y(x = x_0) = y_0 \tag{39}$$

Методы численного решения уравнения (38) могут быть разделены на явные и неявные. Методы также отличаются друг от друга точностью и устойчивостью. Наиболее распространенными методами решения можно назвать методы Рунге-Кутта, Эйлера, BDF.

Простейшим методом является явный метод Эйлера:

$$y_{i+1} = y_i + hf(x_i, y_i) \tag{40}$$

Создайте функцию

```
double euler(func1d f, double y, double x, double dx)
```

реализующую один шаг метода Эйлера. Тип func1d — это указатель на правую часть уравнения $f(x, y)$.

Методом следующего порядка точности $O(h^2)$ является метод

$$y_{i+1} = y_i + hf\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}f(x_i, y_i)\right) \tag{41}$$

Создайте функцию

```
double midpoint(func1d f, double y, double x,
                double dx)
```

реализующую один шаг этого метода.

В качестве тестового примера можно выбрать уравнение с

$$f(x, y) = ay + b \tag{42}$$

с начальным условием $y(0) = 0$. *Сравните результаты метода Эйлера и метода второго порядка с точным аналитическим решением.* Для этого удобно результаты решений выводить в файл, из которого потом строятся решения $y(x)$. Для построения графиков можно использовать программу gnuplot. Пусть данные в файле output.dat храняться в формате x, y_1, y_2. Тогда, чтобы нарисовать кривые $y_1(x)$ и $y_2(x)$, нужно использовать команду

```
plot 'output.dat' u 1:2 w l, 'output.dat' u 1:3 w p
```

10 Системы ОДУ

Решение систем дифференциальных уравнений наталкивается на трудности, которые не встречаются при решении одного уравнения. Очень часто системы, получающиеся на практике, являются жесткими. Это свойство проще всего видеть на примере линейных уравнений:

$$\frac{d\mathbf{y}}{dt} = A\mathbf{y} \tag{43}$$

с начальным условием $\mathbf{y}(\mathbf{0}) = \mathbf{y_0}$. Если все собственные значения матрицы A различны, то существует матрица M, такая то $MAM^{-1} = \Lambda$ — диагональная матрица, состоящая из собственных значений матрицы A. Переходя к переменной $\mathbf{z} = M\mathbf{y}$, имеем

$$\frac{d\mathbf{z}}{dt} = \Lambda\mathbf{z}$$

Решение этой системы

$$z_i = z_i(0)\exp(\lambda_i t)$$

где λ_i — собственные значения матрицы A.

Метод Эйлера для системы (43) с шагом h имеет вид:

$$\mathbf{y_{n+1}} = (1 + hA)\mathbf{y_n}$$

Можно показать, что для сходимости этого метода необходимо выполнение условия $h|\lambda_i| < 2$ для всех собственных значений. Если разброс величин λ_i большой, это требование приводит к очень жестким ограничениям на шаг. Такая система называется жесткой системой ОДУ. Заметим, что собственные значения не соответствуют, вообще говоря, скорости изменения y_i. Поэтому, при решении жестких систем возникает неприятная ситуация, когда требуется все время уменьшать шаг, хотя решение может меняться довольно медленно. Уменьшение шага ведет к увеличении времени расчета и, что более важно, к росту вычислительной погрешности. В конце концов оказывается, что заданную систему вообще не возможно решить выбранным методом, поскольку требуются слишком малые значения шага.

Определение жесткости для нелинейных систем дать довольно сложно. На практике, жесткость такой системы проявляется именно как необходимость выбора все меньших и меньших шагов. Можно показать, что все явные линейные многошаговые методы не удовлетворяют необходимым требованиям устойчивости и, следовательно, не подходят для решения жестких систем [3].

Поэтому для решения жестких систем разработаны неявные методы, в которых значение $\mathbf{y}$ на следующем шаге находится из решения уравнений, в общем случае, нелинейных. Наиболее употребительными методами такого класса являются методы BDF (Backward Differentiation Formulae). Приведем соответствующие соотношения [4], для методов BDF порядка 1 – 5:

$$y_{n+1} - y_n = hf_{n+1} \tag{44}$$

$$3y_{n+2} - 4y_{n+1} + y_n = 2hf_{n+2} \tag{45}$$

$$11y_{n+3} - 18y_{n+2} + 9y_{n+1} - 2y_n = 6hf_{n+3} \tag{46}$$

$$25y_{n+4} - 48y_{n+3} + 36y_{n+2} - 16y_{n+1} + 3y_n = 12hf_{n+4} \tag{47}$$

$$\begin{aligned} 137y_{n+5} - 300y_{n+4} + 300y_{n+3} - 200y_{n+2} + 75y_{n+1} - \\ -12y_n = 60hf_{n+5} \end{aligned} \tag{48}$$

Система ОДУ предполагаются записанной в виде (38), $f_{n+k} = f(t_{n+k}, y_{n+k})$. Первое из этих уравнений, определяющее метод первого порядка, называется также неявным методом Эйлера. Методы порядка выше 5 не употребимы так как всегда неустойчивы.

При использовании BDF нахождение значения y на новом шаге требует решения системы нелинейных уравнений. Чтобы не усложнять задачу, ограничимся решением линейной системы (43). *Напишите программу, демонстрирующую решение такой системы уравнений методами BDF первого и второго порядков.* В качестве тестового примера можно рассмотреть систему [4]:

$$A = \begin{pmatrix} -8003 & 1999 \\ 23988 & -6004 \end{pmatrix} \tag{49}$$

с начальным условием

$$y(0) = \begin{pmatrix} 1 \\ 4 \end{pmatrix} \tag{50}$$

Собственные значения равны $\lambda_1 = -7$ и $\lambda_2 = -14000$. Решение системы $y_1 = \exp(-7x)$, $y_2 = 4\exp(-7x)$. *Убедитесь в том, что при достаточно большом шаге, $h = 0.004$, явный*

метод Эйлера расходится, в то время как неявный дает решение. Сравните точность решения, получаемого этим методом и методом BDF второго порядка.

11 Системы нелинейных уравнений

Требуется решить систему уравнений

$$F_i(x_1, x_2, \dots x_n) \equiv \mathbf{F}(\mathbf{x}) = 0 \tag{51}$$

$i = 1 \dots n$. Такая задача встречается достаточно часто, например, при использовании неявных методов решения ОДУ, решении задач оптимизации и др. Однако, в отличие от решения одного нелинейного уравнения, здесь предложить какой-либо достаточно универсальный алгоритм, приводящий к успеху, практически невозможно. Это связано с тем, что функции F_i, вообще говоря, могут быть никак друг с другом не связаны, так что уравнения могут иметь несколько решений, расположенных в различных областях [1]. Все методы, используемые для решения, оказываются очень чувствительны к выбору начальной точки, от которой выдуться итерации.

Простейшим способом является обобщение метода простой итерации. Можно напрямую использовать соотношения (36), в которых вместо скалярной величины x подставляется вектор $\mathbf{x}$, либо итерировать компоненты вектора $\mathbf{x}$ последовательно. При этом условие сжимаемости, необходимое для сходимости простых итераций, выполнить становится сложнее, чем для одного уравнения.

Более универсальным является многомерный вариант метода Ньютона. Разложение функции $\mathbf{F}$ вблизи точки $\mathbf{x}$ имеет вид:

$$\mathbf{F}(\mathbf{x} + \tilde{\mathbf{x}}) = \mathbf{F}(\mathbf{x}) + \mathbf{J}\tilde{\mathbf{x}} \tag{52}$$

где $\mathbf{J}$ — якобиан, вычисленный в точке $\mathbf{x}$:

$$J_{ik} = \frac{\partial F_i}{\partial x_k} \tag{53}$$

Поэтому поправка к решению, $\tilde{\mathbf{x}}$, находится из решения системы линейных уравнений:

$$\mathbf{J}\tilde{\mathbf{x}} = -\mathbf{F} \tag{54}$$

Особенностью этого метода является необходимость задания якобиана $\mathbf{J}$. Во многих практических случаях получить явное аналитическое выражение для якобиана не удается. В таком случае можно воспользоваться численной аппроксимацией в конечных разностях:

$$J_{ik} = \frac{F_i(x_k + h) - F_i(x_k)}{h} \tag{55}$$

За величину приращения h можно принять εx_k, где ε — заранее выбранное число, одинаковое для всех x_k.

Реализовать в виде функции `newt` *многомерный метод Ньютона. Зависимости* $\mathbf{F}$ *и функция, задающая якобиан,* $\mathbf{J}$*, должны передаваться как параметры. Также создать функцию* `jacob` *, осуществляющую численное вычисление якобиана. На тестовых примерах сравнить эффективность метода Ньютона с аналитическим и численным заданием якобиана.*

В качестве примера системы уравнений можно взять следующую систему [5]:

$$0.5\sin(xy) - 0.25y/\pi - 0.5x = 0 \tag{56}$$

$$(1 - 0.25/\pi)(\exp(2x) - e) + ey/\pi - 2ex = 0. \tag{57}$$

У этих уравнений известны два решения: $(0.29945, 2.83693)$ и $(0.5, 3.14159)$.

12 Моделирование физических процессов

В этой главе сформулированы физические задачи, для решения которых используются изложенные выше численные методы.

12.1 Теплопроводность

Решить уравнение теплопроводности

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t} \tag{58}$$

с граничными условиями $u\,(x=1)=0$, $u\,(x=0)=1$ и начальным условием $u\,(t=0)=0$. Конечно-разностная схема для этого уравнения может быть записана так:

$$\frac{1}{h^2}\left(u_{i-1} - 2u_i + u_{i+1}\right) = \frac{u_i - \hat{u}_i}{\tau}$$

где $u_i = u\,(x=x_i)$ — новое значение температуры, а $\hat{u}$ — значение температуры на предыдущем шаге по времени, h — шаг по координате, τ — шаг по времени. Таким образом, новые значения температуры находятся решением трехдиагональной системы уравнений. Для решения воспользоваться методом прогонки.

Для проверки нужно обратить внимание на то, что со временем температура должна выходить на стационарное значение. Так как в этом случае $\partial u/\partial t = 0$, то из (58) $\partial^2 u/\partial x^2 = 0$, то есть зависимость $u(x)$ — прямая линия.

12.2 Разложение первого порядка

Разложение вещества определяется реакцией первого порядка

$$\frac{dc}{dt} = -c\exp\left(-1/u\right), \tag{59}$$

где c — концентрация, $u = T/E$ — безразмерная температура, нормированная на энергию связи E . Считая, что температура линейно зависит от времени, $u = u_0 + \alpha t$, построить зависимость $c\,(t)$ и dc/dt . Начальное условие $c\,(0)=1$.

С ростом температуры скорость разложения увеличивается, однако при больших временах она должна стремиться к нулю, так как вещество заканчивается. Поэтому зависимость dc/dt от времени должна иметь максимум. В этом можно убедиться и непосредственно, продифференцировав (59).

12.3 Рассеяние в центральном поле

Частица налетает на рассеивающий центр с потенциалом $U\,(r) = -\alpha/r^k$, $k \geq 1$. Прицельный параметр ρ , энергия $E = mv_\infty^2/2$. Считая движение инфинитным, найти угол поворота частицы $\chi = |\pi - 2\phi_0|$, где

$$\phi_0 = \int\limits_{\zeta_{\min}}^{\infty} \frac{\zeta^{-2} d\zeta}{\sqrt{1 - \zeta^{-2} - U/E}}$$

$\zeta = r/\rho$,$\zeta_{\min}$ - точка, в которой знаменатель обращается в ноль. Чтобы не решать нелинейное уравнение для поиска точки поворота, удобно интегрировать от «бесконечности».

Для проверки можно воспользоваться известным результатом для кулоновского поля, $U = -\alpha/r$. В этом случае интеграл вычисляется аналитически [6].

12.4 Точка остановки

Частица налетает на рассеивающий центр с потенциалом $U(r) = -\alpha/r^k$, $k \geq 1$. Прицельный параметр ρ , энергия $E = mv_\infty^2/2$. Наименьшее расстояние до центра определяется решением уравнения

$$1 - \zeta^{-2} + \beta\zeta^{-k} = 0$$

, $\zeta = r/\rho$. Найти это наименьшее расстояние.

Для проверки воспользоваться решением для кулоновского поля, $U = -\alpha/r$.

13 Лабораторная работа 1. Неявные методы решения ОДУ.

Уравнение для решения

$$y' = y^2 + g(x), \quad y(0) = 2 \tag{60}$$

где

$$g(x) = \frac{-x^4 + 6x^3 - 12x^2 + 14x - 9}{(1+x)^2} \tag{61}$$

Точное решение этого уравнения

$$y(x) = \frac{(1-x)(2-x)}{1+x} \tag{62}$$

Задание

- Записать в файл точное решение уравнения (2 балла).
- Решить уравнение (60) методом Эйлера (2 балла).
- Решить уравнение (60) методом, приведенном в варианте ниже. Сравнить решение с решением методом Эйлера. Уравнение, получающееся для нахождения решения на следующем шаге при применении неявного метода, решить несколькими способами: а) воспользовавшись явным видом функции $f(x, y)$ для данной задачи (60) (6 баллов); б) методом Ньютона, выбирая в качестве начальной точки значение y, полученное на предыдущем шаге (5 баллов); в) методом Ньютона, выбирая в качестве начальной точки значение, найденное явным методом Эйлера, $y_{i+1} = y_i + hf_i$ (5 баллов).

Во всех вариантах $f_k \equiv f(x_k, y_k)$.

Вариант 1

Решить одношаговым неявным методом (правило трапеций)

$$y_{i+1} = y_i + \frac{h}{2}(f_i + f_{i+1}) \tag{63}$$

Вариант 2

Неявный метод Эйлера

$$y_{i+1} = y_i + hf_{i+1} \tag{64}$$

Вариант 3

Решить уравнение неявным линейным трехшаговым методом (формула Симпсона)

$$y_{i+1} = y_{i-1} + \frac{h}{3}(f_{i-1} + 4f_i + f_{i+1}) \tag{65}$$

В качестве значений y_0, y_1 использовать точное решение (62).

Вариант 4

Неявный метод Адамса-Мутона

$$y_{i+2} = y_{i+1} + \frac{h}{12}(5f_{i+2} + 8f_{i+1} - f_i) \tag{66}$$

В качестве первых точек y_0, y_1 использовать точное решение (62).

Вариант 5

Реализовать неявный метод Адамса-Мутона (Adams-Moulton)

$$y_{i+3} = y_{i+2} + \frac{h}{24}(9f_{i+3} + 19f_{i+2} - 5f_{i+1} + f_i) \tag{67}$$

В качестве значений y_0, y_1, y_2 использовать точное решение (62).

Вариант 6

Решить уравнение неявным методом BDF 2-го порядка

$$3y_{i+2} - 4y_{i+1} + y_i = 2hf_{i+2} \tag{68}$$

В качестве значений y_0, y_1 использовать точное решение (62).

14 Лабораторная работа 2. Системы нелинейных уравнений

Требуется решить систему уравнений

$$F_i(x_1, x_2, \dots x_n) \equiv \mathbf{F}(\mathbf{x}) = 0 \tag{69}$$

$i = 1 \dots n$. Такая задача встречается достаточно часто, например, при использовании неявных методов решения ОДУ, решении задач оптимизации и др. Однако, в отличие от решения одного нелинейного уравнения, здесь предложить какой-либо достаточно универсальный алгоритм, приводящий к успеху, практически невозможно. Это связано с тем, что функции F_i, вообще говоря, могут быть никак друг с другом не связаны, так что уравнения могут иметь несколько решений, расположенных в различных областях. Все методы, используемые для решения, оказываются очень чувствительны к выбору начальной точки, от которой выдуться итерации.

Простейшим способом является обобщение метода простой итерации, для которого система должна быть записана в виде $\mathbf{x} = \mathbf{g}(\mathbf{x})$. Можно напрямую использовать соотношения $\mathbf{x}_{n+1} = \mathbf{g}(\mathbf{x}_n)$, либо итерировать компоненты вектора $\mathbf{x}$ последовательно. При этом условие сжимаемости, необходимое для сходимости простых итераций, выполнить становится сложнее, чем для одного уравнения.

Более универсальным является многомерный вариант метода Ньютона. Разложение функции $\mathbf{F}$ вблизи точки $\mathbf{x}$ имеет вид:

$$\mathbf{F}(\mathbf{x} + \tilde{\mathbf{x}}) = \mathbf{F}(\mathbf{x}) + \mathbf{J}\tilde{\mathbf{x}} \tag{70}$$

где $\mathbf{J}$ — якобиан, вычисленный в точке $\mathbf{x}$:

$$J_{ik} = \frac{\partial F_i}{\partial x_k} \tag{71}$$

Поэтому поправка к решению, $\tilde{\mathbf{x}}$, находится из решения системы линейных уравнений:

$$\mathbf{J}\tilde{\mathbf{x}} = -\mathbf{F} \tag{72}$$

Особенностью этого метода является необходимость задания якобиана $\mathbf{J}$. Во многих практических случаях получить явное аналитическое выражение для якобиана не удается. В таком случае можно воспользоваться численной аппроксимацией в конечных разностях:

$$J_{ik} = \frac{F_i(x_k + h) - F_i(x_k)}{h} \tag{73}$$

За величину приращения h можно принять εx_k, где ε — заранее выбранное число, одинаковое для всех x_k.

1. (2 балла) Решить систему методом простых итераций. 2. (4 балла) Решить систему методом Ньютона с аналитическим заданием якобиана. 3) (4 балла) То же, с численным якобианом.

Тестовые системы уравнений и их решения:

Вариант 1

$$0.5\sin(xy) - 0.25y/\pi - 0.5x = 0$$
$$(1 - 0.25/\pi)(\exp(2x) - e) + ey/\pi - 2ex = 0.$$

Здесь e — основание натурального лагорифма. Есть два решения: $(0.29945, 2.83693)$ и $(0.5, 3.14159)$.

Вариант 2

$$10^4 xy - 1 = 0$$
$$\exp(-x) + \exp(-y) - 1.001 = 0$$

Решение: $(1.450 \times 10^{-5}, 6.8933353)$.

Вариант 3

$$2x_1 + x_2 + x_3 + x_4 + x_5 = 6$$
$$x_1 + 2x_2 + x_3 + x_4 + x_5 = 6$$
$$x_1 + x_2 + 2x_3 + x_4 + x_5 = 6$$
$$x_1 + x_2 + x_3 + 2x_4 + x_5 = 6$$
$$x_1 x_2 x_3 x_4 x_5 = 1$$

Решения: $(1, 1, 1, 1, 1)$, $(0.916, 0.916, 0.916, 0.916, 1.418)$.

Вариант 4

$$4x_1^3 + 4x_1 x_2 + 2x_2^2 - 42x_1 = 14$$
$$4x_2^3 + 2x_1^2 + 4x_1 x_2 - 26x_2 = 22$$

Эта система имеет несколько решений на промежутке $-5 \leq x_1, x_2 \leq 5$. Некоторые из них: $(3.0, 2.0)$, $(-2.8051, 3.1313)$

Вариант 5

$$x^2 + y^2 = 10$$
$$2x + y = 1$$

Решения: $(-1, 3)$, $(9/5, -13/5)$.

Варинат 6

$$2x^2 + y^2 = 24$$
$$x^2 - y^2 = -12$$

Решения: $(\pm 2, \pm 4)$

Список литературы

[1] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992. ISBN 0-521-43108-5.

[2] Н. С. Бахвалов, Н.П. Жидков, and Г. М. Кобельков. *Численные методы*. БИНОМ. Лаборатория знаний, Москва, 2007. 636 pp.

[3] Germund G. Dahlquist. A special stability problem for linear multistep methods. *BIT Numerical Mathematics*, 3(1):27–43, 1963.

[4] E. Süli and D.F. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, 2003. ISBN 9780521007948. LCCN 2003273860.

[5] C.A. Floudas, P.M. Pardalos, C. Adjiman, W.R. Esposito, Z.H. Gümüs, S.T. Harding, J.L. Klepeis, C.A. Meyer, and C.A. Schweiger. *Handbook of Test Problems in Local and Global Optimization*. Nonconvex Optimization and Its Applications. Springer US, 2013. ISBN 9781475730401.

[6] Л. Д. Ландау and Е. М. Лифшиц. *Теоретическая физика. Том I. Механика*. ФИЗМАТЛИТ, 2007. 224 pp.