Московский авиационный институт
(Национальный исследовательский университет)
Факультет прикладной математики и физики
Кафедра математической кибернетики

Лабораторная работа № 4

по курсу «Численные методы»

Тема: численные методы решения обыкновенных дифференциальных уравнений.

Студент: Мукин Ю. Д.
Группа: 80-304Б-18
Преподаватель: Гидаспов В.Ю.

Москва, 2021

Задание 1

1) Постановка задачи: Реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки h . С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

Вариант 12:

Задача Коши:

$$(x^2+1)y''-2xy'+2y=0,$$
$$y(0)=1,$$
$$y'(0)=1,$$
$$x\in[0,1], h=0.1$$

Точное решение:

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

2) Теория :

Неявный метод Эйлера

Если на правой границе интервала использовать точное значение производной от решения (т.е. тангенса угла наклона касательной), то получается неявный метод Эйлера первого порядка точности.

$$y_{k+1}=y_k+hf(x_{k+1},y_{k+1}) \qquad (4.3)$$

В общем случае нелинейное относительно y_{k+1} уравнение (4.3) численно решается с помощью одного из методов раздела 2, например, методом Ньютона или его модификациями.

Метод Рунге-Кутты четвертого порядка $(p=4, a_1=0, a_2=\frac{1}{2}, a_3=\frac{1}{2}, a_4=1, b_{21}=\frac{1}{2}, b_{31}=0, b_{32}=\frac{1}{2}, b_{41}=0, b_{42}=0, b_{43}=\frac{1}{2}, c_1=\frac{1}{6},$
$c_2=\frac{1}{3}, c_3=\frac{1}{3}, c_3=\frac{1}{6})$

является одним из самых широко используемых методов для решения Задачи Коши:

$$y_{k+1} = y_k + \Delta y_k$$
$$\Delta y_k = \frac{1}{6}(K_1^k + 2K_2^k + 2K_3^k + K_4^k) \quad (4$$

$$K_1^k = hf(x_k, y_k)$$
$$K_2^k = hf(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_1^k)$$
$$K_3^k = hf(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_2^k)$$
$$K_4^k = hf(x_k + h, y_k + K_3^k)$$

Метод Адамса

При использовании интерполяционного многочлена 3-ей степени построенного по значениям подынтегральной функции в последних четырех узлах получим метод Адамса четвертого порядка точности:

$$y_{k+1} = y_k + \frac{h}{24}(55f_k - 59f_{k-1} + 37f_{k-2} - 9f_{k-3}), \quad (4.25)$$

где f_k значение подынтегральной функции в узле x_k.

Метод Адамса (4.25) как и все многошаговые методы не является самостартующим, то есть для того, что бы использовать метод Адамса необходимо иметь решения в первых четырех узлах. В узле x_0 решение y_0 известно из начальных условий, а в других трех узлах x_1, x_2, x_3 решения y_1, y_2, y_3 можно получить с помощью подходящего одношагового метода, например: метода Рунге-Кутты четвертого порядка

3) Результат работы программы:

Eiler_method______________

k	x	y	error
0	0.0000000	1.0000000	0.0000000
1	0.1000000	1.1000000	0.1000000
2	0.2000000	1.1800000	0.0900000
3	0.3000000	1.2398020	0.0798020
4	0.4000000	1.2792117	0.0692117
5	0.5000000	1.2980422	0.0580422
6	0.6000000	1.2961159	0.0461159

```
|  7| 0.7000000| 1.2732668| 0.0332668|
|  8| 0.8000000| 1.2293411| 0.0193411|
|  9| 0.9000000| 1.1641973| 0.0041973|
| 10| 1.0000000| 1.0777061| 0.0122939|
_________Runge_Kut_method_____________
| k |    x    |     y    |   error  |
|  0| 0.0000000| 1.0000000| 0.0000000|
|  1| 0.1000000| 1.0900000| 0.0000000|
|  2| 0.2000000| 1.1600000| 0.0000000|
|  3| 0.3000000| 1.2100001| 0.0000001|
|  4| 0.4000000| 1.2400004| 0.0000004|
|  5| 0.5000000| 1.2500007| 0.0000007|
|  6| 0.6000000| 1.2400011| 0.0000011|
|  7| 0.7000000| 1.2100016| 0.0000016|
|  8| 0.8000000| 1.1600023| 0.0000023|
|  9| 0.9000000| 1.0900030| 0.0000030|
| 10| 1.0000000| 1.0000038| 0.0000038|
____________Adams_method______________
| k |    x    |     y    |   error  |
|  0| 0.0000000| 1.0000000| 0.0000000|
|  1| 0.1000000| 1.1000000| 0.0100000|
|  2| 0.2000000| 1.1800000| 0.0200000|
|  3| 0.3000000| 1.2398020| 0.0298020|
|  4| 0.4000000| 1.2792117| 0.0392117|
|  5| 0.5000000| 1.2876772| 0.0376772|
|  6| 0.6000000| 1.2751104| 0.0351104|
|  7| 0.7000000| 1.2424068| 0.0324068|
|  8| 0.8000000| 1.1883764| 0.0283764|
|  9| 0.9000000| 1.1138203| 0.0238203|
| 10| 1.0000000| 1.0185146| 0.0185146|
```

4) Код программы:

```
void delta_y(void(*F)(double x, double yv[], double res[]), double x, double y[], double h)
{
   double *tmp = malloc(2*sizeof(double));
```

```
    double *k1 = malloc(2*sizeof(double));
    F(x, y, k1);

    double *k2 = malloc(2*sizeof(double));
    tmp[0] = k1[0]*h*0.5+y[0]; tmp[1] = k1[1]*h*0.5+y[1];
    F(x + h/2, tmp, k2);

    double *k3 = malloc(2*sizeof(double));
    tmp[0] = k2[0]*h*0.5+y[0]; tmp[1] = k2[1]*h*0.5+y[1];
    F(x + h/2, tmp, k3);

    double *k4 = malloc(2*sizeof(double));
    tmp[0] = k3[0]*h+y[0]; tmp[1] = k3[1]*h+y[1];
    F(x + h, tmp, k4);
    y[0] += (k1[0]+k4[0]+k2[0]*2+k3[0]*2)*(h/6); y[1] += (k1[1]+k4[1]+k2[1]*2+k3[1]*2)*(h/6);
    free(tmp); free(k1); free(k2); free(k3); free(k4);
}

void F_v12_1(double x, double yv[], double res[])
{
    double z = yv[1];
    double y = yv[0];
    res[0] = z;
    res[1] = (2*x*z-2*y) / (pow(x,2)+1);
}

void get_start_value_v12_1(double y[])
{
    y[0] = 1;
    y[1] = 1;
}

double precise_v12_1(double x)
{
    return x - pow(x,2)+1;
}

void Eiler_method(void(*get_start_value)(double y[]), void(*F)(double x, double yv[], double res[]), double(*precise)
(double x), double x0, double b, double h)
{
    int k=0;
    double *y = malloc(2*sizeof(double));
    double *res = malloc(2*sizeof(double));
    get_start_value(y);
    double y_prec = y[0];
    double error = 0;
    printf("\n\n____________Eiler_method______________\n| k |     x    |     y    |   error  |");
    for(double x=x0; x<=b; x+=h)
    {
        printf("\n________________________________________\n|%3d|%10.7lf|%10.7lf|%10.7lf|",k, x, y[0], error);
        k++;
        F(x,y, res);
        y[0] += res[0]*h; y[1] += res[1]*h;
        error = fabs(precise(x) - y[0]);
    }
    free(res); free(y);
}

double Runge_Kut_method(void(*get_start_value)(double y[]), void(*F)(double x, double yv[], double res[]),
double(*precise)(double x), double x0, double b, double h)
{
        int k=0;
        double *y = malloc(2*sizeof(double));
```

```
    get_start_value(y);
            double error = 0;
    printf("\n\n_________Runge_Kut_method_____________\n| k |     x    |     y    |   error  |");
            for(double x=x0; x<=b; x+=h)
            {
        error = fabs(precise(x) - y[0]);
                        printf("\n________________________________________\n|%3d|%10.7lf|%10.7lf|%10.7lf|",k,  x,  y[0], error);
                        k++;
                        delta_y(F, x, y, h);
            }
    error = y[0];
    free(y);
    return error;
}

void  Adams_method(void(*get_start_value)(double  y[]),  void(*F)(double  x,  double  yv[],  double  res[]), double(*precise)(double x), double x0, double b, double h)
{
    double **f = malloc(((int)((b-x0)/h))*sizeof(double*));
            int k=0;
            double *y = malloc(2*sizeof(double));
    double *a1 = malloc(2*sizeof(double));
    double *a2 = malloc(2*sizeof(double));
    double *a3 = malloc(2*sizeof(double));
    double *a4 = malloc(2*sizeof(double));
    get_start_value(y);
    double error = 0;
    double x;
            printf("\n\n____________Adams_method______________\n| k |     x    |     y    |   error  |");
            for(x=x0; k<4; x+=h)
            {
        error = fabs(precise(x) - y[0]);
                        printf("\n________________________________________\n|%3d|%10.7lf|%10.7lf|%10.7lf|",k,  x,  y[0], error);
        f[k] = malloc(2*sizeof(double));
                        F(x,y, f[k]);
        y[0] += f[k][0]*h; y[1] += f[k][1]*h;
        k++;
            }
            for(; x<=b; x+=h)
            {
                        error = fabs(precise(x) - y[0]);
                        printf("\n________________________________________\n|%3d|%10.7lf|%10.7lf|%10.7lf|",k,  x,  y[0], error);
                        f[k] = malloc(2*sizeof(double));
                        F(x,y, f[k]);
                        a1[0] = f[k][0]*55; a1[1] = f[k][1]*55;
                        a2[0] = f[k-1][0]*(-59); a2[1] = f[k-1][1]*(-59);
                        a3[0] = f[k-2][0]*37; a3[1] = f[k-2][1]*37;
                        a4[0] = f[k-3][0]*(-9); a4[1] = f[k-3][1]*(-9);
                        k++;
        y[0] += (a1[0]+a2[0]+a3[0]+a4[0])*(h/24); y[1] += (a1[1]+a2[1]+a3[1]+a4[1])*(h/24);
            }
    free(f); free(y); free(a1); free(a2); free(a3); free(a4);
}

void task_4_1()
{
    double start, end, step;
    printf("specify line boundaries and step size: ");
    scanf("%lf%lf%lf", &start, &end, &step);
    Eiler_method(get_start_value_v12_1, F_v12_1, precise_v12_1, start, end, step);
```

```
  Runge_Kut_method(get_start_value_v12_1, F_v12_1, precise_v12_1, start, end, step);
  Adams_method(get_start_value_v12_1, F_v12_1, precise_v12_1, start, end, step);
}
```

Задание 2

1) Постановка задачи: Реализовать метод стрельбы и конечно-разностный метод решения краевой задачи для ОДУ в виде программ. С использованием разработанного программного обеспечения решить краевую задачу для обыкновенного дифференциального уравнения 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

Вариант 12:

Краевая задача:	Точное решение:
$x(x-1)y''-xy'+y=0$ $y'(1)=3$ $y(3)-3y'(3)=-4$	$y(x)=2+x+2x\ln\lvert x\rvert$

2) Теория :

Суть метода заключена в многократном решении задачи Коши для приближенного нахождения решения краевой задачи.

Пусть надо решить краевую задачу (4.28), (4.29) на отрезке $[a,b]$. Вместо исходной задачи формулируется задача Коши с уравнением (4.28) и с начальными условиями

$$\begin{aligned} y(a) &= y_0 \\ y'(b) &= \eta \end{aligned}, \qquad (4.32)$$

где η - некоторое значение тангенса угла наклона касательной к решению в точке $x = a$.

Положим сначала некоторое начальное значение параметру $\eta = \eta_0$, после чего решим каким либо методом задачу Коши (4.28),(4.32). Пусть $y = y_0(x, y_0, \eta_0)$ решение этой задачи на интервале $[a,b]$, тогда сравнивая значение функции $y_0(b, y_0, \eta_0)$ со значением y_1 в правом конце отрезка можно получить информацию для корректировки угла наклона касательной к решению в левом конце отрезка. Решая задачу Коши для нового значения $\eta = \eta_1$, получим другое решение со значением $y_1(b, y_0, \eta_1)$ на правом конце. Таким образом, значение решения на правом конце $y(b, y_0, \eta)$ будет являться функцией одной переменной η. Задачу можно сформулировать таким образом: требуется найти такое значение переменной η^*, чтобы решение $y(b, y_0, \eta^*)$ в правом конце отрезка совпало со значением y_1 из (4.29). Другими словами решение исходной задачи эквивалентно нахождению корня уравнения

$$\Phi(\eta) = 0\,, \qquad (4.33)$$

где $\Phi(\eta) = y(b, y_0, \eta) - y_1$.

Рассмотрим двухточечную краевую задачу для линейного дифференциального уравнения второго порядка на отрезке $[a,b]$

$$y'' + p(x)y' + q(x)y = f(x) \qquad (4.35)$$

$$y(a) = y_0, y(b) = y_1 \ . \qquad (4.36)$$

Введем разностную сетку на отрезке $[a,b]$ $\Omega^{(h)} = \{x_k = x_0 + hk\}$, $k = 0,1,....,N$, $h = |b-a|/N$. Решение задачи (4.35),(4.36) будем искать в виде сеточной функции $y^{(h)} = \{y_k, k = 0,1,...., N\}$, предлагая, что решение существует и единственно. Введем разностную аппроксимацию производных следующим образом:

$$y_k' = \frac{y_{k+1} - y_{k-1}}{2h} + O(h^2)\,;$$

$$y_k'' = \frac{y_{k+1} - 2y_k + y_{k-1}}{h^2} + O(h^2)\,; \qquad (4.37)$$

Подставляя аппроксимации производных из (4.37) в (4.35),(4.36) получим систему уравнений для нахождения y_k:

$$\begin{cases} y_0 = y_a \\ \dfrac{y_{k+1} - 2y_k + y_{k-1}}{h^2} + p(x_k)\dfrac{y_{k+1} - y_{k-1}}{2h} + q(x_k)y_k = f(x_k), k = \overline{1, N-1} \\ y_N = y_b \end{cases} \quad (4.38)$$

Приводя подобные и учитывая, что при задании граничных условий первого рода два неизвестных y_0, y_N уже фактически определены, получим систему линейных алгебраических уравнений с трехдиагональной матрицей коэффициентов

$$\begin{cases} (-2 + h^2 q(x_1))y_1 + (1 + \dfrac{p(x_1)h}{2})y_2 = h^2 f(x_1) - (1 - \dfrac{p(x_1)h}{2})y_a \\ (1 - \dfrac{p(x_k)h}{2})y_{k-1} + (-2 + h^2 q(x_k))y_k + (1 + \dfrac{p(x_k)h}{2})y_{k+1} = h^2 f(x_k) \qquad ,k=2,...,N\text{-}2 \\ (1 - \dfrac{p(x_{N-1})h}{2})y_{N-1} + (-2 + h^2 q(x_{N-1}))y_{N-1} = h^2 f(x_{N-1}) - (1 + \dfrac{p(x_{N-1})h}{2})y_b \end{cases} \quad (4.39)$$

Для системы (4.39) при достаточно малых шагах сетки h и $q(x_k) < 0$ выполнены условия преобладания диагональных элементов

$$\left|-2 + h^2 q(x_k)\right| > \left|1 - \frac{p(x_k)h}{2}\right| + \left|1 + \frac{p(x_k)h}{2}\right|, \quad (4.39)$$

что гарантирует устойчивость счета и корректность применения метода прогонки для решения этой системы.

3) Результат работы программы:

enter the desired precision: 0.1

________finite_differences________
| x | y |y precise|
____
| 1.1000000| 3.3102564| 3.3096824|
____
| 1.2000000| 3.6391608| 3.6375717|
____
| 1.3000000| 3.9851593| 3.9821471|
____
| 1.4000000| 4.3469368| 4.3421223|
____
| 1.5000000| 4.7233664| 4.7163953|
____
| 1.6000000| 5.1134711| 5.1040116|

1.7000000	5.5163964	5.5041361
1.8000000	5.9313881	5.9160320
1.9000000	6.3577757	6.3390448
2.0000000	6.7949596	6.7725887
2.1000000	7.2423999	7.2161368
2.2000000	7.6996082	7.6692124
2.3000000	8.1661405	8.1313820
2.4000000	8.6415914	8.6022499
2.5000000	9.1255894	9.0814537
2.6000000	9.6177924	9.5686595
2.7000000	10.1178850	10.0635596
2.8000000	10.6255750	10.5658687
2.9000000	11.1405909	11.0753223
3.0000000	11.5926803	11.5916737

__________shoot_methods__________

x	y	y precise
1.0000000	3.1747886	3.0000000
1.1000000	3.4747886	3.3096824
1.2000000	3.8047886	3.6375717
1.3000000	4.1488987	3.9821471
1.4000000	4.5065310	4.3421223
1.5000000	4.8769921	4.7163953
1.6000000	5.2595947	5.1040116
1.7000000	5.6536913	5.5041361
1.8000000	6.0586832	5.9160320
1.9000000	6.4740210	6.3390448
2.0000000	6.8992019	6.7725887
2.1000000	7.3337653	7.2161368
2.2000000	7.7772891	7.6692124
2.3000000	8.2293853	8.1313820
2.4000000	8.6896968	8.6022499
2.5000000	9.1578938	9.0814537

____
| 2.6000000| 9.6336713| 9.5686595|

____
| 2.7000000|10.1167462|10.0635596|

____
| 2.8000000|10.6068554|10.5658687|

____
| 2.9000000|11.1037538|11.0753223|

____
| 3.0000000|11.6002122|11.5916737|

4) Код программы:

```
double f(double x, double y, double z)
{
   return z;
}

double g(double x, double y, double z)
{
   if((x*x-x)==0) return 3;
   return (x*z-y)/(x*x-x);
}

double shoot(double ksi, double coef[], int N, double ya, double yb, double h, double x[], double y[], double z[])
{
   y[0] = ksi;
   z[0] = (ya-coef[0]*ksi)/coef[1];
   for(int i=0; i<=N; i++)
   {
      y[i+1] = y[i] +h*f(x[i], y[i], z[i]);
      z[i+1] = z[i] +h*g(x[i], y[i], z[i]);
   }
return coef[2]*z[N-1] + coef[3]*y[N-1]-yb;
}

void shoot_method(double coef[], double a, double b, double h, double eps)
{
   int N = (int)((b-a)/h) + 1;
   double ya = 3, yb = -4;
   double *x = malloc(N*sizeof(double));
   double *y = malloc(N*sizeof(double));
   double *z = malloc(N*sizeof(double));
   for(int i=0; i<=N; i++)
      x[i] = a+h*i;
   double teta0 = 2, teta1 =3;
    double teta = teta1 - ((teta1 - teta0)*shoot(teta1, coef, N, ya, yb, h, x, y, z) / (shoot(teta1, coef, N, ya, yb, h, x, y, z) - shoot(teta0, coef, N, ya, yb, h, x, y, z)));
   while(fabs(shoot(teta, coef, N, ya, yb, h, x, y, z)) >= eps)
   {
      teta0 = teta1;
      teta1=teta;
         teta = teta1 - ((teta1 - teta0)*shoot(teta1, coef, N, ya, yb, h, x, y, z) / (shoot(teta1, coef, N, ya, yb, h, x, y, z) - shoot(teta0, coef, N, ya, yb, h, x, y, z)));
   }
   shoot(teta, coef, N, ya, yb, h, x, y, z);
   printf("\n\n__________shoot_methods__________\n|     x    |     y    |y  precise|");
   for(int i=0; i<=N; i++)
      printf("\n________________________________\n|%10.7lf|%10.7lf|%10.7lf|",x[i], y[i], precise_v12_2(x[i]));
   free(x); free(y); free(z);
}
```

```c
double finite_differences(double(*value)(double x), double(*p)(double x), double(*q)(double x), double(*f)(double x), double yb[], double a, double b, double h)/*yb[] - массив с 2 коэффицентами: 1 - множител при производной в выражении для точки, 2 - чему равно выражение в точке*/
{
    matrix A, B;
    double x = a+h;
    int N = (int)(fabs(b-a)/h);
    A = create_matrix(N+1,N+1); B = create_matrix(1,N+1);
    *get_element(&A, 0, 0) = -2 + h*h*q(x); *get_element(&A, 0, 1) = 1+p(x)*h/2; *get_element(&B, 0, 0) = h*h*f(x) - (1-p(x)*h/2)*value(a);
    x+=h;
    for(int i=1; i<N; i++, x+=h)
    {
        *get_element(&A, i, i-1) = 1-p(x)*h/2;
        *get_element(&A, i, i) = -2 + h*h*q(x);
        *get_element(&A, i, i+1) = 1+p(x)*h/2;
        *get_element(&B, 0, i) = h*h*f(x);
    }
    *get_element(&A, N, N-1) = -yb[0]; *get_element(&A, N, N) = yb[0]+h; *get_element(&B, 0, N) = yb[1]*h;
    matrix X = run_method(A, B, N+1);
    x=a+h;
    printf("\n\n________finite_differences________\n|     x    |     y    |y  precise|");
    for(int i=0; i<=N; i++)
    {
        printf("\n________________________________\n|%10.7lf|%10.7lf|%10.7lf|", x, X.body[i], precise_v12_2(x));
        x+=h;
    }
}

void task_4_2()
{
    double start, end, step, eps;
    double *y = malloc(2*sizeof(double));
    y[0] = -3; y[1] = -4;
    printf("specify line boundaries and step size: ");
    scanf("%lf%lf%lf", &start, &end, &step);
    printf("enter the desired precision: ");
    scanf("%lf", &eps);
    double *coef = malloc(4*sizeof(double));
    coef[0] = 0; coef[1] = 1; coef[2] = -3; coef[3] = 1;
    finite_differences(precise_v12_2, p_func_v12, q_func_v12, f_func_v12, y, start, end, step);
    shoot_method(coef, start, end, step, eps);
    free(y); free(coef);
    //1 3 0.1 0.1
}
```