

## Отчет по лабораторной работе 5 Кобак Ф.А. 18ДКК-1

### явный Метод Эйлера

```
% expr – выражение через которую выражается первая производная
% x0 – начальный x, он же начало отрезка на котором ищется решение
% y0 – начальный y
% b – конец отрезка на котором ищут решение
% h – шаг
% info – отображать ли отладочную информацию
% resX, resY – массивы с решениями

function [resX , resY] = eilerMetod(expr , x0 , y0 , b , h , info)

    % инициализируем начальный y
    resY = y0;
    % x заполнится весь в цикле потому инициализируем пустой массив
    resX = [];

    % счетчик итераций
    counter = 1;

    for( Xcount = x0:h:b)
        % заполнение массива x
        resX = [resX , Xcount];

        % высчитываем значение функции выражающей производную в текущей точке
        fVal = subs(expr , {'x' 'y'} , {resX(counter) , resY(counter)});
        % по формуле ищем следующий y
        resY(counter + 1) = resY(counter) + fVal*h;

        % отладочная информация ++++++
        if(info)
            disp(['iteration number ' , num2str(counter) , '++++++'])
            disp([num2str(resY(counter)) , ' + ' , num2str(h) , '*' , num2str(fVal)
, ' = ' , num2str(resY(counter + 1))]);
            disp(['iteration number ' , num2str(counter) , '++++++'])
        end
        % отладочная информация ++++++
        % к счетчику доплюсовываем итерацию
        counter = counter + 1;

    end
    % так построен цикл, что получается один лишний y, потому его выкинем из
    результата
    resY(numel(resY)) = [];

end
```

### неявный Метод Эйлера

```
% expr – выражение через которую выражается первая производная
% x0 – начальный x, он же начало отрезка на котором ищется решение
```

```

% y0 – начальный y
% b – конец отрезка на котором ищут решение
% h – шаг
% info – отображать ли отладочную информацию
% resX, resY – массивы с решениями
function [resX , resY] = eilerMetodSpesh(expr , x0 , y0 , b , h, info)

    % инициализируем начальный y
    resY = y0;
    % x заполнится весь в цикле потому инициализируем пустой массив
    resX = [];

    % объявляем переменную с помощью которой будем описывать уравнение для каждой
    % итерации
    syms y;
    % получаем имена переменных используемые в переданном выражении
    vars = symvar(expr);

    % счетчик итераций
    counter = 1;

    % перебираем требуемый массив x
    for( Xcount = x0:h:b)

        % заполнение массива x
        resX = [resX , Xcount];

        % подставляем в выражение производной следующий x (resX(counter) + h)
        % и переменную y, таким образом формируется выражение через которое
        % выражается resY(i+1)
        f_xi_yi = subs(expr , {vars} , {resX(counter) + h , y});
        % решаем уравнение  $y(i+1) = y(i) + h \cdot f(x(i+1), y(i+1))$ 
        resY(counter + 1) = solve(resY(counter) + h*f_xi_yi - y);

        % отладочная информация+++++++
        if(info)
            disp(['iteration number ' , num2str(counter) , '++++++'])
            disp('equation is')
            disp(['y = ' , num2str(resY(counter)) , ' + ' , num2str(h) , '*( ' ,
char(f_xi_yi) , ')']);
            disp(['y = ' , num2str(resY(counter + 1))]);
            disp(['iteration number ' , num2str(counter) , '++++++'])
        end
        % отладочная информация+++++++
        % к счетчику доплюсовываем итерацию
        counter = counter + 1;

    end

    % так построен цикл, что получается один лишний y, потому его выкинем из
    % результата
    resY(numel(resY)) = [];

end

```

## Метод Хойна

```

% expr – выражение через которую выражается первая производная
% x0 – начальный x, он же начало отрезка на котором ищется решение
% y0 – начальный y

```

```

% b – конец отрезка на котором ищут решение
% h – шаг
% info – отображать ли отладочную информацию
% resX, resY – массивы с решениями

```

```

function [resX , resY] = hoinMetod(expr , x0 , y0 , b , h, info)

    % инициализируем начальный y
    resY = y0;
    % x заполнится весь в цикле потому инициализируем пустой массив
    resX = [];

    % счетчик итераций
    counter = 1;

    for( Xcount = x0:h:b)

        % заполнение массива x
        resX = [resX , Xcount];
        % находим значение переданной функции, в текущей точке
        fVal = subs(expr , {'x' 'y'} , {resX(counter) , resY(counter)});
        %  $f(x(i) + h, y(i) + (h/2)*f(x(i), y(i)))$ 
        fVal2 = subs(expr , {'x' , 'y'} , {resX(counter) + h , resY(counter) +
h*fVal});

        % далее по формуле находим новый y
        resY(counter + 1) = resY(counter) + (h/2)*(fVal + fVal2);

        % отладочная информация+++++++
        if(info)
            disp(['iteration number ' , num2str(counter) , '++++++'])
            disp([num2str(resY(counter)) , ' + (' , num2str(h) , '/2) * (' ,
num2str(fVal) , ' + f(' , num2str(resX(counter) + h) , ', ' , num2str(resY(counter)+1)
, ' + ' , num2str(h) , '* ' , num2str(fVal) , ') = ']);
            disp([num2str(resY(counter)) , ' + ' , num2str(h/2) , ' * (' ,
num2str(fVal) , ' + ' , num2str(fVal2) , ') = ' , num2str(resY(counter + 1))]);
            disp(['iteration number ' , num2str(counter) , '++++++'])
        end
        % отладочная информация+++++++
        % к счетчику доплюсовываем итерацию
        counter = counter + 1;

    end

    % так построен цикл, что получается один лишний y, потому его выкинем из
    результата
    resY(numel(resY)) = [];

end

```

Метод Адамса (делал по [https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4\\_%D0%90%D0%B4%D0%B0%D0%BC%D1%81%D0%B0](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%90%D0%B4%D0%B0%D0%BC%D1%81%D0%B0))

```

% expr – выражение через которую выражается первая производная
% x0 – начальный x, он же начало отрезка на котором ищется решение
% y0 – начальный y
% b – конец отрезка на котором ищут решение
% h – шаг
% info – отображать ли отладочную информацию

```

```

% x1 , y1 – вторая точка для метода

function [rx , ry] = adamsMetod(expr , x0 , y0 , b , x1 , y1, h, info)

    % инициализируем начальный x, все остальные значения заполню в цикле
    rx = x0 ;

    % инициализируем начальный y, теми значениями, которые известны
    ry = [y0 , y1];

    % начинаю счет с 2 так, как 2 точки уже известны
    counter = 2;

    % для первой итерации, искусственно создаю “значение функции на предыдущей
    итерации”
    fVal = subs(expr , {'x' 'y'} , {rx(counter-1) ry(counter-1)});

    % перебираю x от второй точки и до конца отрезка, на котором требуется найти
    решение
    for (x = x1:h:b)
        % заполняем массив результатов
        rx = [rx , x];

        % запоминаем значение функции на предыдущей итерации
        fValP = fVal;
        % вычисляем значение функции для текущей итерации
        fVal = subs(expr , {'x' 'y'} , {rx(counter) , ry(counter)});

        % по формуле вычисляем новый y
        ry(counter + 1) = ry(counter) + (h/2)*(3*fVal - fValP);

        % отладочная информация+++++++
        if(info)
            disp(['iteration ' , num2str(counter), '++++++'])
            disp([num2str(ry(counter)) , ' + (' , num2str(h) , '/2)*(' ,
num2str(3*fVal) , ' - ' , num2str(fValP),') = ' , num2str(ry(counter + 1))]);
            disp(['iteration ' , num2str(counter), '++++++'])
        end
        % отладочная информация+++++++

        % к счетчику доплюсовываем итерацию
        counter = counter + 1;

    end

    % обнуляем последний ну – он лишний
    ry(numel(ry)) = [];

end

```

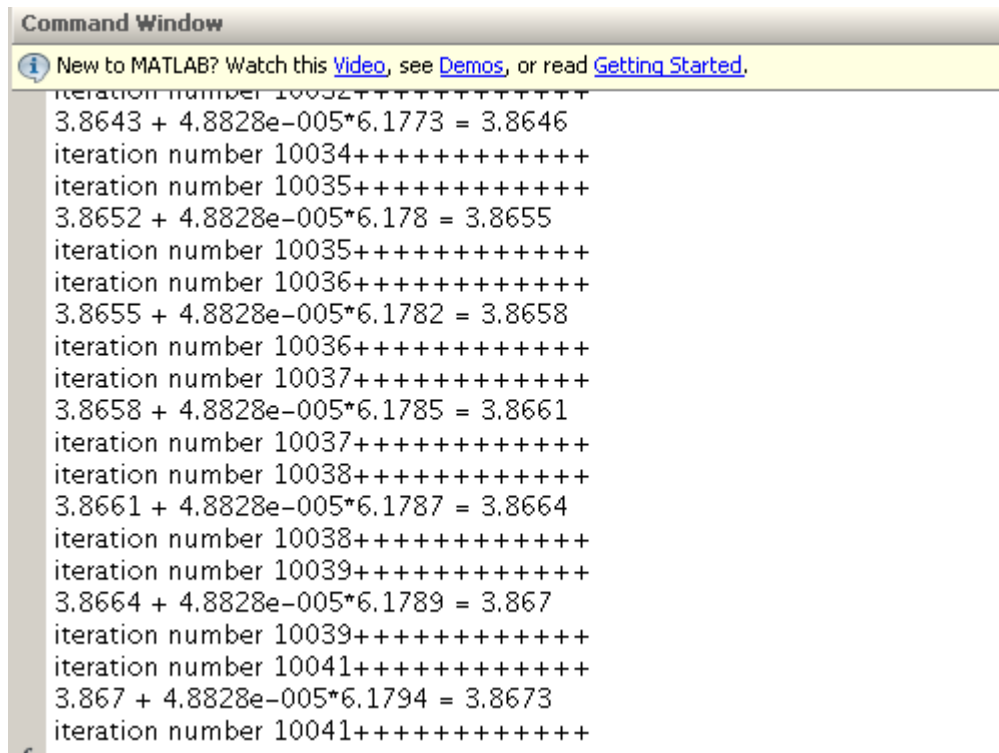
## Результирующий скрипт

```

% указываю функцию с которой буду работать
f = sym('y/(x^2) + exp(x)');
% остальные параметры
x0 = 1; y0 = 1.3679; h = 0.1; b=2;epsil = 0.6737;% не такая точность, дело в том,

```

% что както получается так, что с заданной точностью  $10^{-4}$  методы эйлера очень  
%долго сходятся, вот я прождал до 100000 итерации, и он и близко до требуемой  
%точности не дошел, потому пришлось понизить точность



The screenshot shows a MATLAB Command Window with a title bar. Below the title bar is a yellow information banner that reads: "New to MATLAB? Watch this [Video](#), see [Demos](#), or read [Getting Started](#)." Below the banner, the command window displays a series of iterative calculations. Each iteration starts with "iteration number" followed by a number, then a plus sign, a value in scientific notation (4.8828e-005), a multiplication sign, another value (6.1773, 6.178, 6.1782, 6.1785, 6.1787, 6.1789, 6.1794), an equals sign, and a final result. The iterations shown are 10032 through 10041. The results are: 3.8646, 3.8655, 3.8658, 3.8661, 3.8664, 3.8667, 3.8673.

```
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
iteration number 10032+++++++
3.8643 + 4.8828e-005*6.1773 = 3.8646
iteration number 10034+++++++
iteration number 10035+++++++
3.8652 + 4.8828e-005*6.178 = 3.8655
iteration number 10035+++++++
iteration number 10036+++++++
3.8655 + 4.8828e-005*6.1782 = 3.8658
iteration number 10036+++++++
iteration number 10037+++++++
3.8658 + 4.8828e-005*6.1785 = 3.8661
iteration number 10037+++++++
iteration number 10038+++++++
3.8661 + 4.8828e-005*6.1787 = 3.8664
iteration number 10038+++++++
iteration number 10039+++++++
3.8664 + 4.8828e-005*6.1789 = 3.8667
iteration number 10039+++++++
iteration number 10041+++++++
3.867 + 4.8828e-005*6.1794 = 3.8673
iteration number 10041+++++++
```

% отменяю вывод информации  
info = false;

% задаю начальный шаг, он будет дробиться по мере приближения к требуемой точности  
tempH = h;  
% первый прогон, шагом увеличим на два, перед циклом, он нужен для оценки  
%точности первой итерации  
[rxh2, ryh2] = eilerMetod(f, x0, y0, b, tempH\*2, info);  
while (true)

    % прогоняем метод  
    [rx, ry] = eilerMetod(f, x0, y0, b, tempH, info);

    % тут я получаю каждый второй элемент решения, дело в том, что для  
    %использования той формулы для оценки точности, что нам давали на лекциях,  
    %требуется высчитывать решение на в два раза более широкой сетке  
    % а в на ней соответственно меньше вектор решения получается, потому размерности  
    % векторов не совпадают.  
    % лучшее что я придумал, это выкинуть те решения которые не лежат на в два раза  
    % более широкой сетке для оценки точности  
    t\_ry = ry(1:2:numel(ry));  
    t\_rx = rx(1:2:numel(rx));

    % тут вычисляю норму разницы между решением на широкой сетке и решением на  
    % узкой сетке  
    % get1Norm – самописная функция, представлена в конце отчета  
    diffR = get1Norm(ryh2 - t\_ry);  
    % проверяю сходимость  
    if(diffR < epsil)

```

        break;% если сошелся покидаю цикл
    end

    rxh2 = rx; ryh2 = ry;% запоминаю значения на этой итерации, ведь они будут
    % совпадать с значениями для широкой сетки в на следующей итерации
    % бью шаг между узлами на два
    tempH = tempH/2;

end

% вывод результата
disp('Euler metod solution');
eulerSol = [rx;ry]
% нарисую интегральную линию для решения этого диффур.
plot(rx , ry , 'r');
hold on;

% для неявного метода Эйлера все аналогично, только вычисления ведутся через
% другую формулу
tempH = h;
[rxh2 , ryh2] = eilerMetodSpesh(f , x0 , y0, b , tempH*2 , info);% отличие 1
while (true)

    [rx, ry] = eilerMetodSpesh(f , x0 , y0, b , tempH , info);% отличие 2

    t_ry = ry(1:2: numel(ry));
    t_rx = rx(1:2: numel(rx));

    % get1Norm – самописная функция, представлена в конце отчета
    diffR = get1Norm(ryh2 - t_ry)

    if(diffR < epsil)
        break;
    end

    rxh2 = rx; ryh2 = ry;
    tempH = tempH/2;

end

% также вывод результата
disp('implicit Euler metod solution');
eilerSpeshSol = [rx;ry]
% интегральная кривая розовым
plot(rx , ry , 'm');
hold on;

% метод Хойна сходиллся куда быстрее потому для него я смог поставить требуемую
точность
epsil = 10^(-4);
% кроме этого опять же все аналогично, кроме критерия сходимости
tempH = h;
[rxh2 , ryh2] = hoinMetod(f , x0 , y0, b , tempH*2 , info);
while (true)

    [rx, ry] = hoinMetod(f , x0 , y0, b , tempH , info);

    t_ry = ry(1:2: numel(ry));

```

```

t_rx = rx(1:2: numel(rx));
% get1Norm – самописная функция, представлена в конце отчета
diffR = get1Norm(ryh2 - t_ry)

%!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! ТУТ /3 т.к. это метод второго порядка
% сходимости
if(diffR/3 < epsil)
    break;
end

rxh2 = rx; ryh2 = ry;
tempH = tempH/2;

end

% вывод результата
disp('hoin metod solution');
hoinSol = [rx;ry]
% интегральная кривая
plot(rx , ry , 'g');
hold on;

% с Методом адамса сложнее
% я не смог 'налету' оценивать точность так как необходимо знать два начальных узла
% а между ними расстояние уменьшается вместе с уменьшением шага!
% потому я просто взял два узла из метода Хойна и пересчитал с тем шагом на котором
% закончил метод хойна

% для широкой сетки пропускаем второй узел, его как бы должен 'перескочить' в два
% раза больший шаг
[rxh2, ryh2] = adamsMetod(f , x0 , y0 , b , rx(3) , ry(3) , 2*tempH , info);
% для узкой сетки скапливаем первую и вторую точку, и соответствующий шаг
[rx , ry] = adamsMetod(f , x0 , y0 , b , rx(2) , ry(2) , tempH , info);

% выкидываем те решения которые не лежат на шикой сетке
t_ry = ry(1:2: numel(ry));
t_rx = rx(1:2: numel(rx));
% get1Norm – самописная функция, представлена в конце отчета
diffR = get1Norm(ryh2 - t_ry); % вычисляем точность с которой подсчитали результат

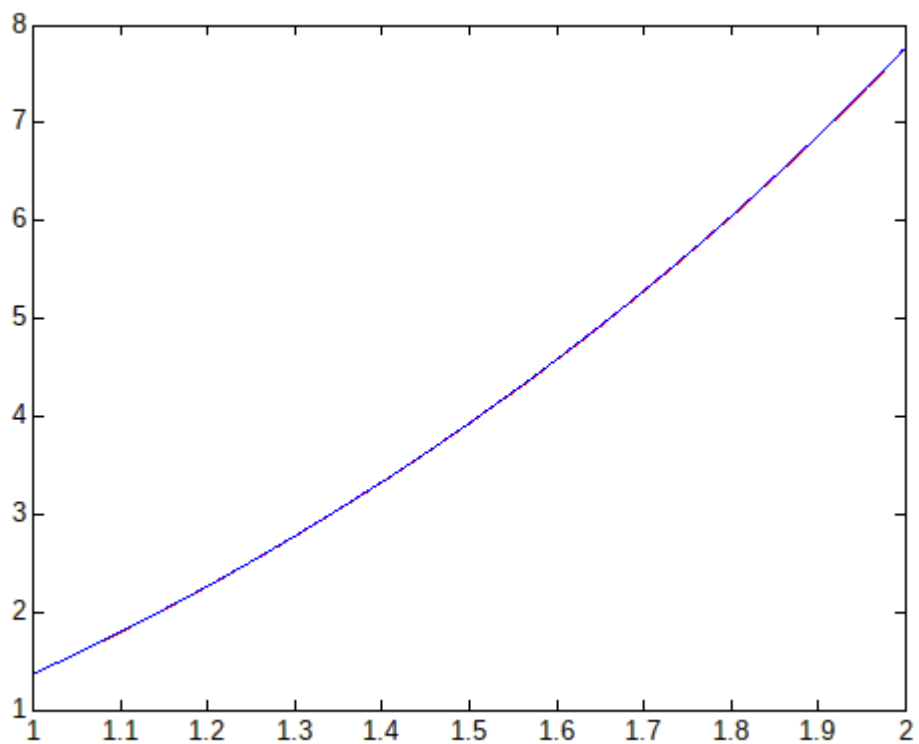
% выводм результат, точность и интегральную линию
disp(['adams metod sulution with accuracy ' , num2str(diffR/3)])
adams = [rx , ry]
plot(rx , ry, 'b');

```

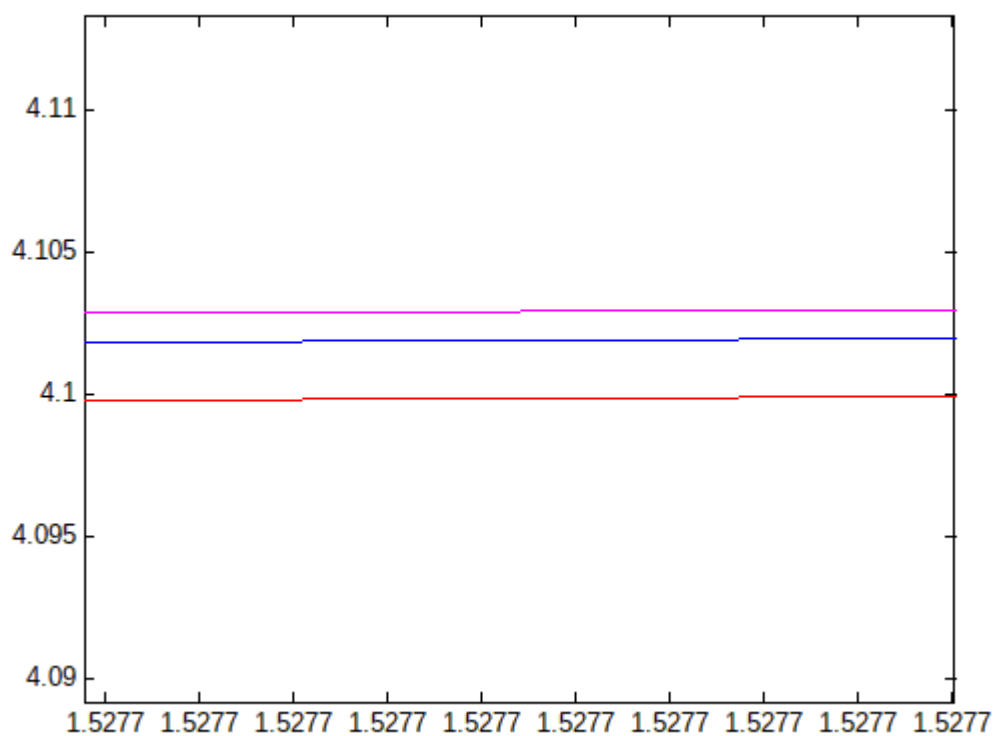
## Результаты выполнения

Получились вектора большой размерности до 5000 значений. Потому я почистил workspace – оставил только эти вектора, сохранил workspace и отправил его вместе с отчетом.

Но на иртегральные кривые можно посмтреть и здесь.

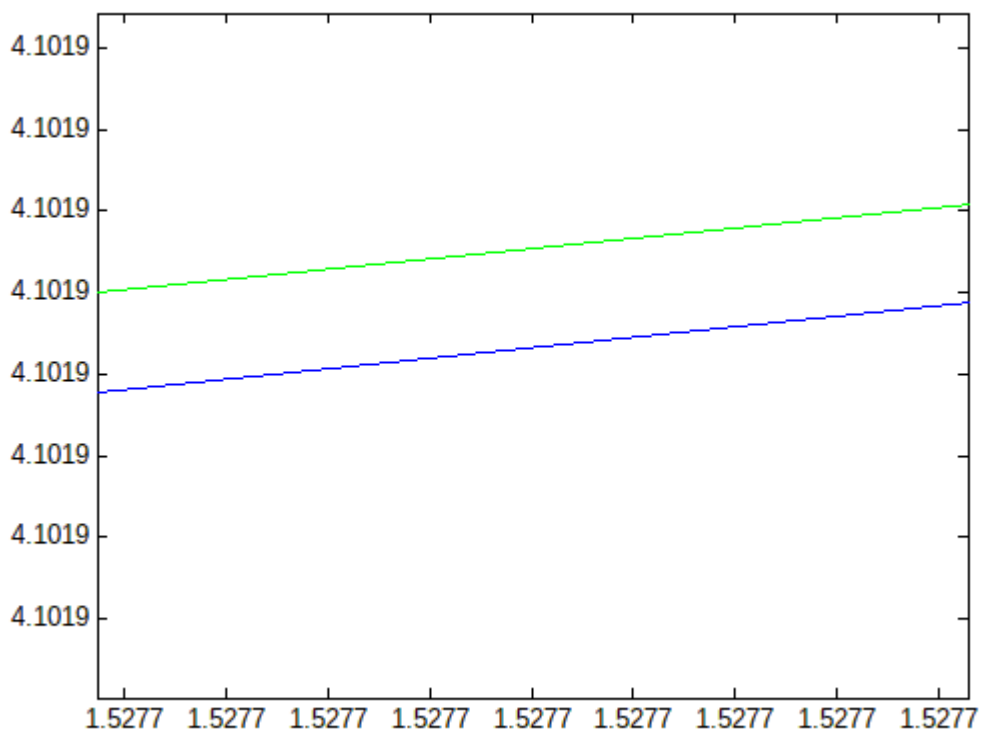


Выглядит как одна, но это потому, что они 'срослись' но при ближайшем рассмотрении





и еще ближе



метод адамса удалось реализовать с точностью  
0.00010551

Вспомогательная функция для получения первой нормы

```
function res = get1Norm(A)

    ASize = size(A);
    sumVec = 0;
    for i = 1:ASize(1)
        sumVec = [sumVec , sum(A(i ,:))];
    end

    res = max(sumVec);
end
```