Санкт-Петербургский *Национальный Исследовательский Университет ИТМО*
Факультет программной инженерии и компьютерной техники

Вычислительная математика

Лабораторная работа №5

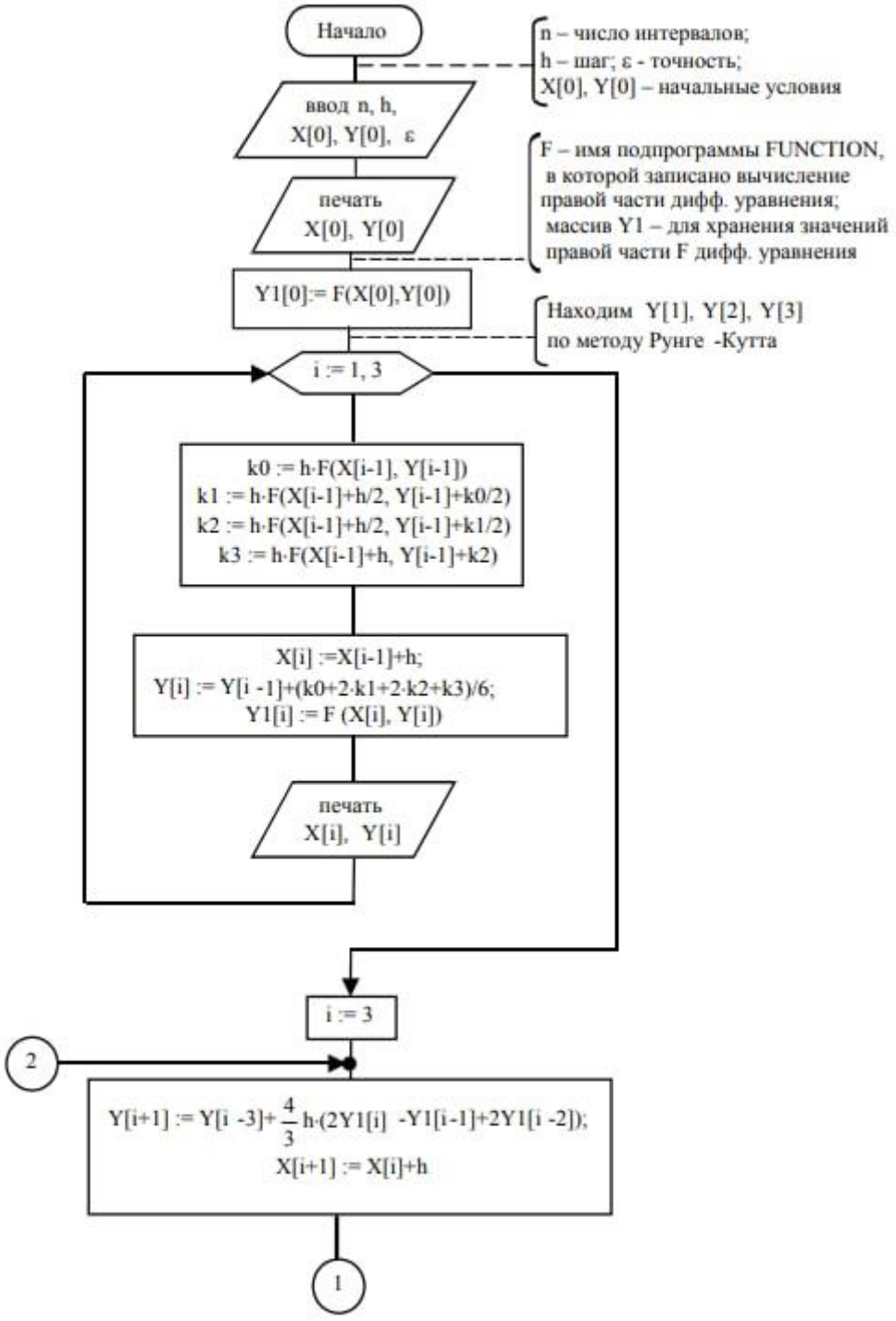
Вариант: метод Милна

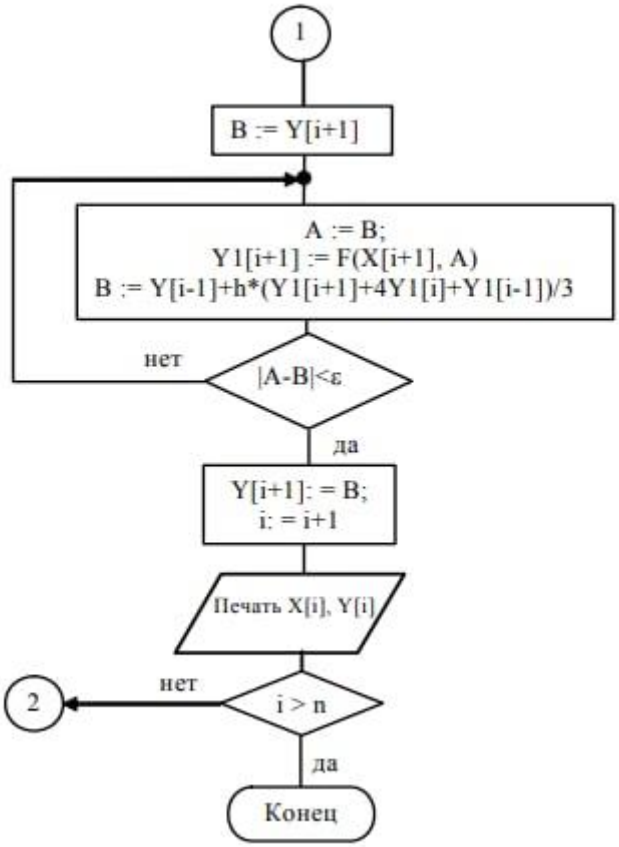
Выполнил:
студент группы P3231
Нестеров Иван Алексеевич

Преподаватель:
Перл Ольга Вячеславовна

г. Санкт-Петербург
2022 г.

Блок-схема указанного метода:





Описание метода:

Суть метода сводится к поиску приближенного значения в два этапа. На первом этапе осуществляется поиск некоторого значения, которое называется прогнозом. Это значение является достаточно грубым приближением. Однако на втором шаге прогноз уточняется и находится другое значение, которое называется коррекцией. Интерполяция в данном алгоритме осуществляется при помощи полинома Ньютона. Однако важно отметить, что для метода Милна необходим начальный отрезок из 4 значений: u_0, u_1, u_2, u_3. Первое из них известно из условия, а другие 3 необходимо найти каким-то другим методом, который, в отличие от метода Милна, будет являться самостартующим. В качестве такого чаще всего применяют метод Рунге-Кутта 4 порядка, поэтому и я решил его применить. Далее скажем о том, что модуль разницы двух полученных приближений, деленный на константу, принято рассматривать как значение погрешности. Для метода Милна за константу берется число 1/29. Если полученная погрешность слишком высока, алгоритм повторяет вышеуказанные действия, пока точность не станет удовлетворительной. Поэтому он называется многошаговым. Поиск значений прогноза и коррекции, смысл которых описан выше, осуществляется по определенным для метода Милна формулам и в коде выглядит следующим

образом:

```
Double yPredict = y.get(i-4) + 4.0 / 3.0 * h * (2 * f.get(i-1) - f.get(i-2) + 2 * f.get(i-3));
Double yCorrect = y.get(i-2) + h / 3.0 * (f(x.get(i), yPredict) + 4 * f.get(i-1) + f.get(i-2));
```

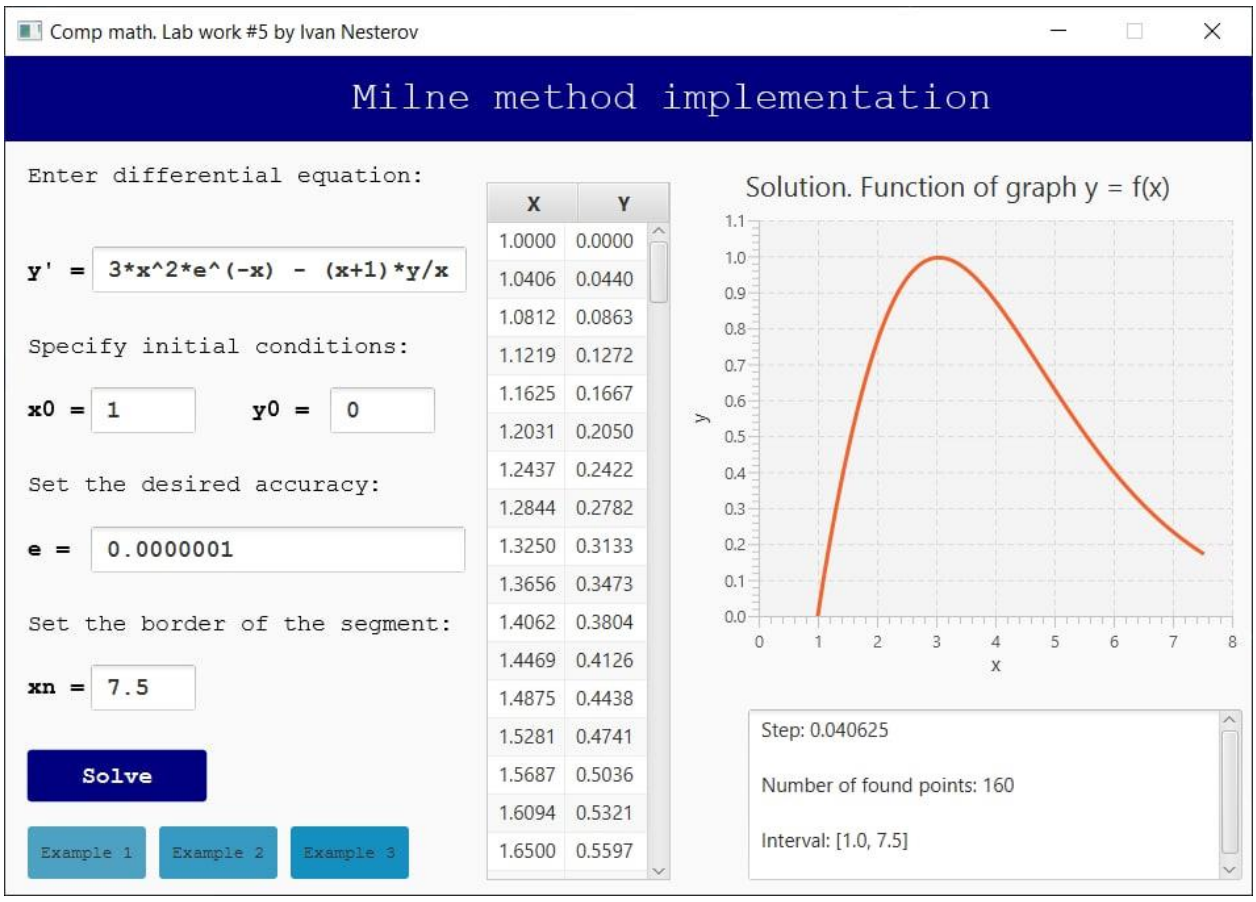
Листинг вычислительного метода:

```
public List<Double> getYList() {
    n = calculateN();
    for (int i = 1; i < n+1; i++){
        if (i == 1 || i == 2 || i ==3){
            y.add(i,getYRungeKutta(i));
            f.add(i,f(x.get(i), y.get(i)));
        }else{
            x.add(i, x.get(i-1)+h);
            Double yPredict = y.get(i-4) + 4.0 / 3.0 * h * (2 * f.get(i-1) -
f.get(i-2) + 2 * f.get(i-3));
            Double yCorrect = y.get(i-2) + h / 3.0 * (f(x.get(i), yPredict) +
4 * f.get(i-1) + f.get(i-2));
            double currentEpsilon = Math.abs(yCorrect - yPredict)/29.0;
            if (currentEpsilon <= epsilon){
                y.add(i, yCorrect);
                f.add(i, f(x.get(i), y.get(i)));
            }else{
                h = h/2.0;
                n = calculateN();
                i = 0;
                Double x0 = x.get(0);
                Double y0 = y.get(0);
                x.clear();
                y.clear();
                f.clear();
                x.add(0, x0);
                y.add(0, y0);
                f.add(0, f(x.get(0), y.get(0)));
            }
        }
    }
    return y;
}
```

Листинг «вспомогательного» вычислительного метода (Рунге-Кутта 4 порядка):

```
public Double getYRungeKutta(int i) {
    x.add(i, x.get(i-1)+h);
    Double k0 = h * f(x.get(i-1), y.get(i-1));
    Double k1 = h * f(x.get(i-1) + h / 2, y.get(i-1) + k0 / 2);
    Double k2 = h * f(x.get(i-1) + h / 2, y.get(i-1) + k1 / 2);
    Double k3 = h * f(x.get(i-1) + h, y.get(i-1) + k2);
    return y.get(i-1) + (k0 + 2 * k1 + 2 * k2 + k3) / 6;
}
```

Демонстрация работы программы:



Для удобства визуализации получаемых результатов на «графическую» часть приложения перенес не только непосредственно график, но и все остальное взаимодейтсвие с пользователем тоже. Добавил заполнение формы тремя придуманными функциями на тот случай, когда не хочется вводить данные самостоятельно.

Вывод:

Comp math. Lab work #5 by Ivan Nesterov

Milne method implementation

Enter differential equation:

y' = 3*x^2*e^(-x) - (x+1)*y/x

Specify initial conditions:

x0 = 1 y0 = 0

Set the desired accuracy:

e = 0.0000001

Set the border of the segment:

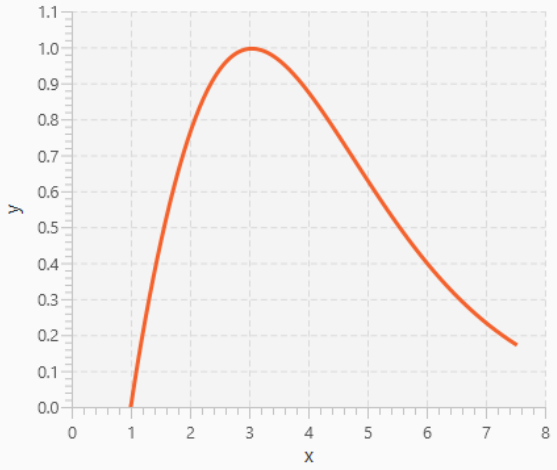
xn = 7.5

Solve

Example 1 Example 2 Example 3

X	Y
1.0000	0.0000
1.0406	0.0440
1.0812	0.0863
1.1219	0.1272
1.1625	0.1667
1.2031	0.2050
1.2437	0.2422
1.2844	0.2782
1.3250	0.3133
1.3656	0.3473
1.4062	0.3804
1.4469	0.4126
1.4875	0.4438
1.5281	0.4741
1.5687	0.5036
1.6094	0.5321
1.6500	0.5597

Solution. Function of graph y = f(x)



Step: 0.040625

Number of found points: 160

Interval: [1.0, 7.5]

Comp math. Lab work #5 by Ivan Nesterov

Milne method implementation

Enter differential equation:

y' = 3*x^2*e^(-x) - (x+1)*y/x

Specify initial conditions:

x0 = 1 y0 = 0

Set the desired accuracy:

e = 0.0001

Set the border of the segment:

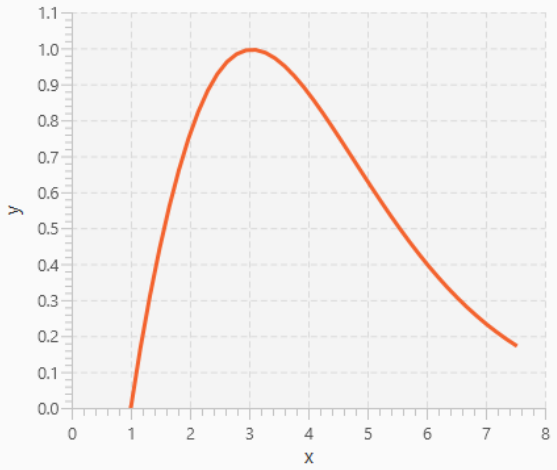
xn = 7.5

Solve

Example 1 Example 2 Example 3

X	Y
1.0000	0.0000
1.1625	0.1667
1.3250	0.3133
1.4875	0.4438
1.6500	0.5598
1.8125	0.6615
1.9750	0.7491
2.1375	0.8225
2.3000	0.8823
2.4625	0.9284
2.6250	0.9621
2.7875	0.9837
2.9500	0.9945
3.1125	0.9954
3.2750	0.9877
3.4375	0.9722
3.6000	0.9505

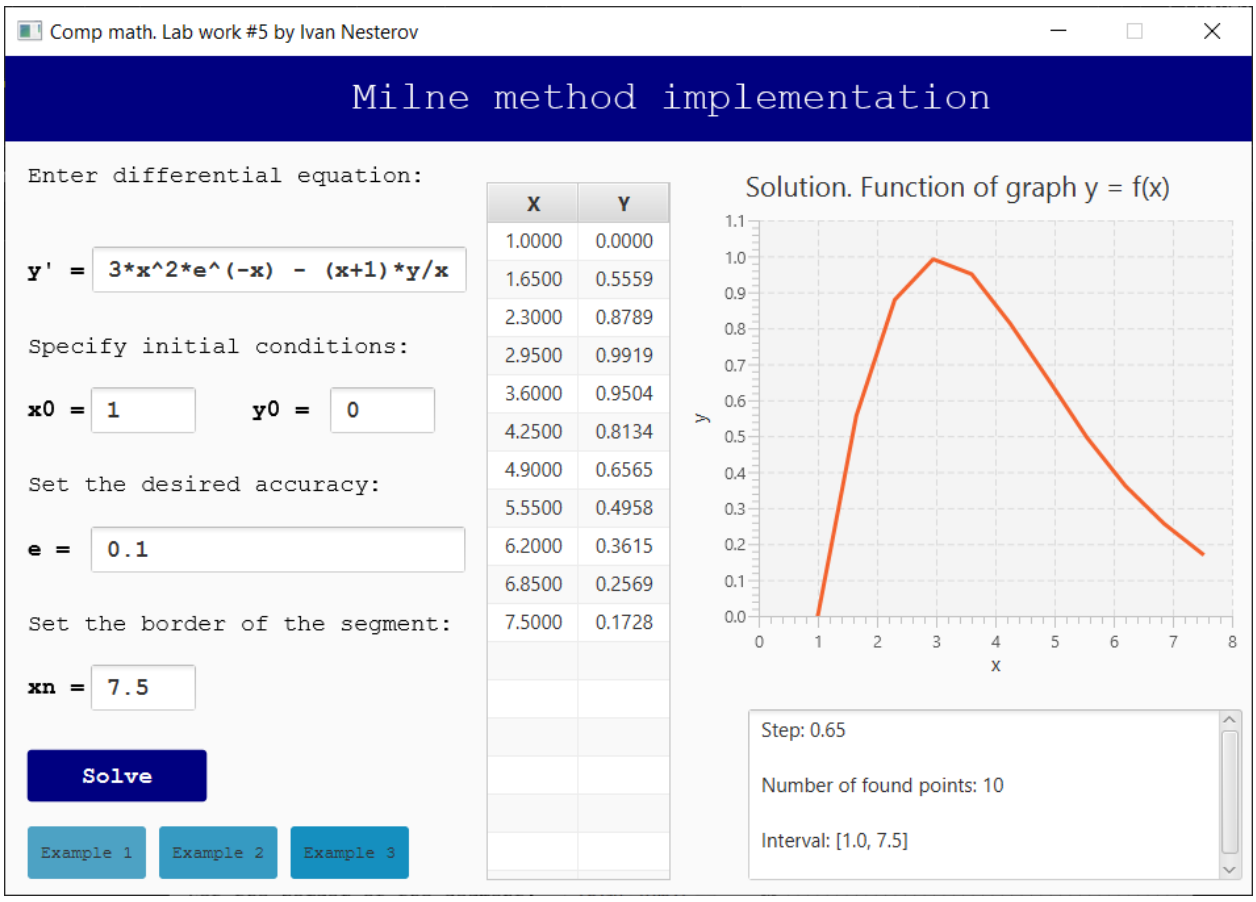
Solution. Function of graph y = f(x)



Step: 0.1625

Number of found points: 40

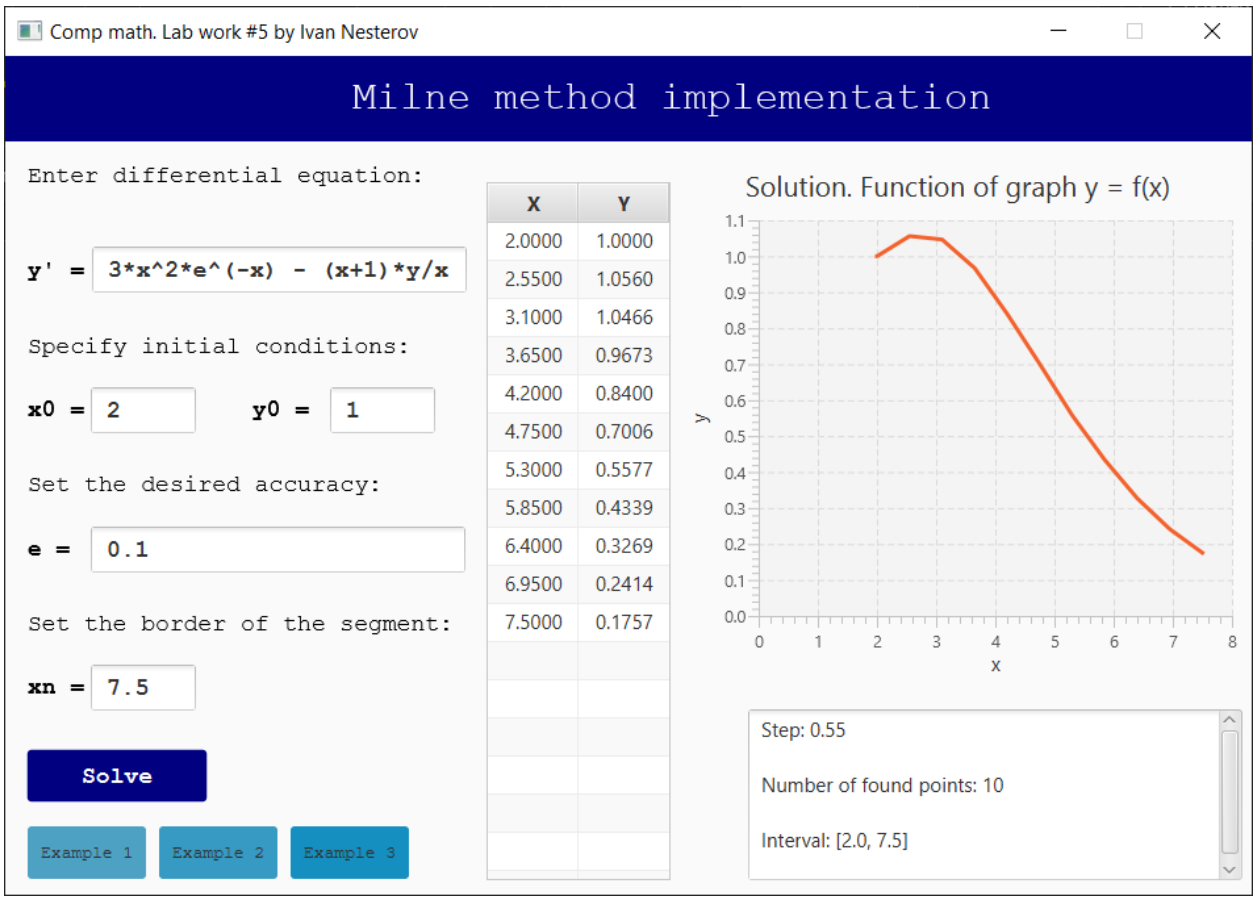
Interval: [1.0, 7.5]



На скриншотах продемонстрирована логичная реакция программы на уменьшение нужной точности – метод раньше завершает свою работу, полученные значения являются приближенными более грубо, а функция, как следствие, выглядит более и более «угловатой». Точек, разумеется, тоже находится меньше и меньше.

Потерю точности вычисления при «понижении планки» у алгоритма могу так же оправдать тем, что значения в таком случае вычисляются быстрее (как минимум потому, что их меньше нужно найти), шаг становится больше, что так же видно на графике. Если для пользователя достаточно получить схематичный вид функции из данного «сложного» дифференциального уравнения, то точностью в некоторой мере считаю возможным пренебречь.

Кроме того, изменение начальных условий меняет внешний вид функции: поиск осуществляется на другом интервале. Приведем пример все той же функции из анализа выше:



Сделанные выводы являются визуально заметным подтверждением описанных в разделе описания алгоритма тезисов о многошаговой сущности работы алгоритма.