Университет ИТМО

Факультет программной инженерии и компьютерных технологий

Лабораторная работа №5 по Вычислительной Математике

Выполнил: Богатов Александр Сергеевич

Группа: Р3233

Вариант: метод Милна

Преподаватель: Перл Ольга Вячеславовна

Санкт-Петербург

2022

Описание математического метода:

Метод Милна является одним из многошаговых методов решения задачи Коши: в таких методах для вычисления следующего значения функции используются результаты k предыдущих шагов. Для получения положения новой точки используются формулы прогноза и коррекции.

Прежде чем применять многошаговый метод, необходимо вычислить первые k точек, в нашем случае 4 точки, вычисляемые методом Рунге-Кутта.

Формула прогноза в методе Милна – формула Милна:

$$y_{i+1} = y_{i-3} + \frac{4}{3}h(2y'_i - y'_{i-1} + 2y_{i-2}) \quad + O(h^5)$$

$$O(h^5) = \frac{28}{90}h^5y^{(5)} - \text{погрешность формулы}$$

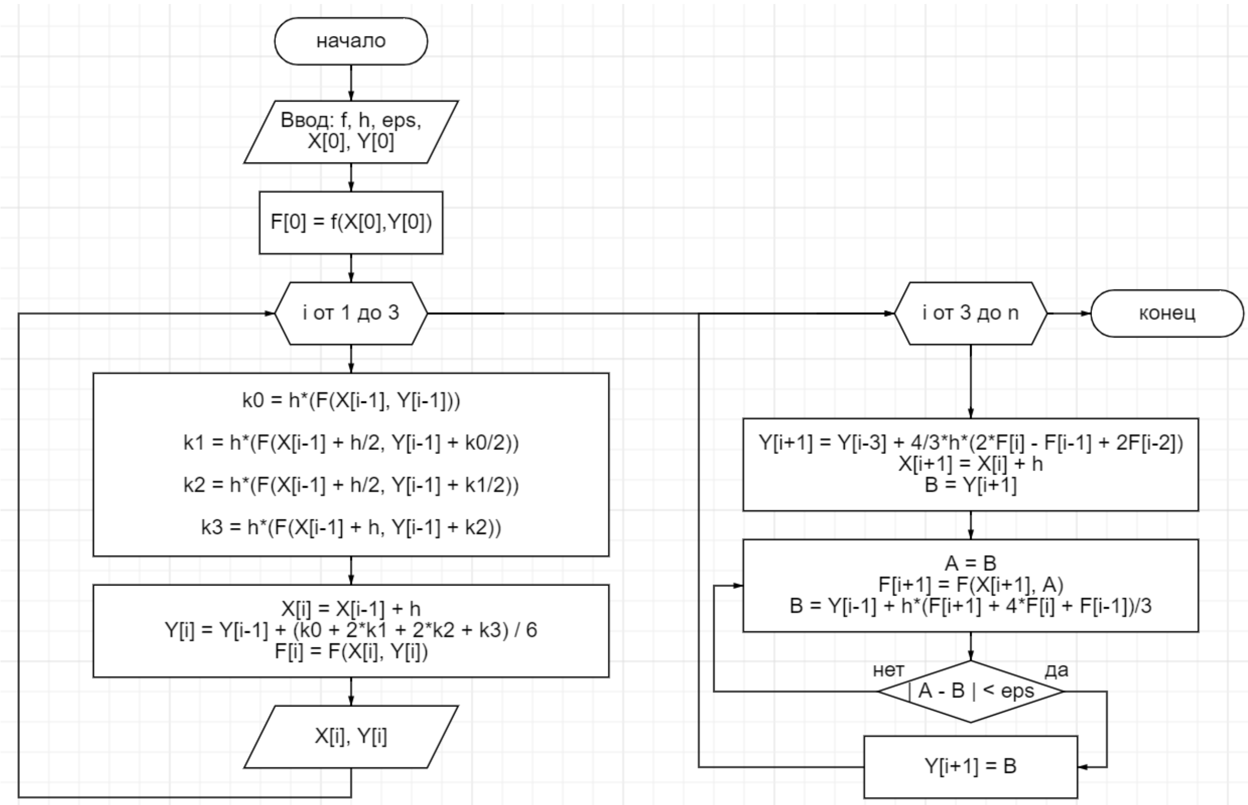
Формула коррекции в методе Милна – формула Симпсона:

$$y_{i+1} = y_{i-1} + \frac{1}{3}h(y'_{i+1} + 4y'_i + y'_{i-1}) \quad + O(h^5)$$

$$O(h^5) = \frac{-1}{90}h^5y^{(5)} - \text{погрешность формулы}$$

Члены содержащие h в пятой или более высоких степенях отбрасываются, в связи с чем метод Милна считается методом четвертого порядка.

Блок-схема математического метода:

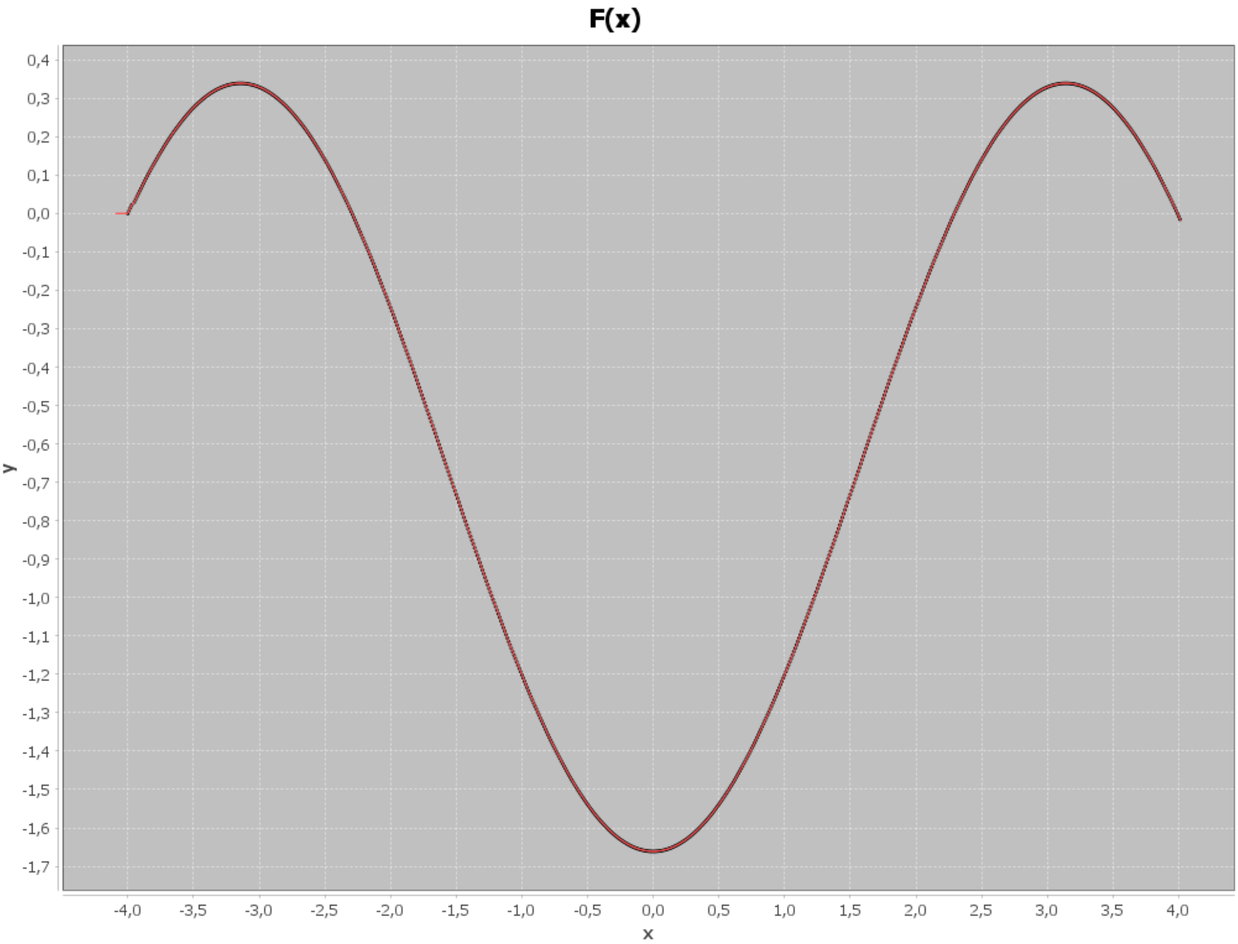


Листинг программы:

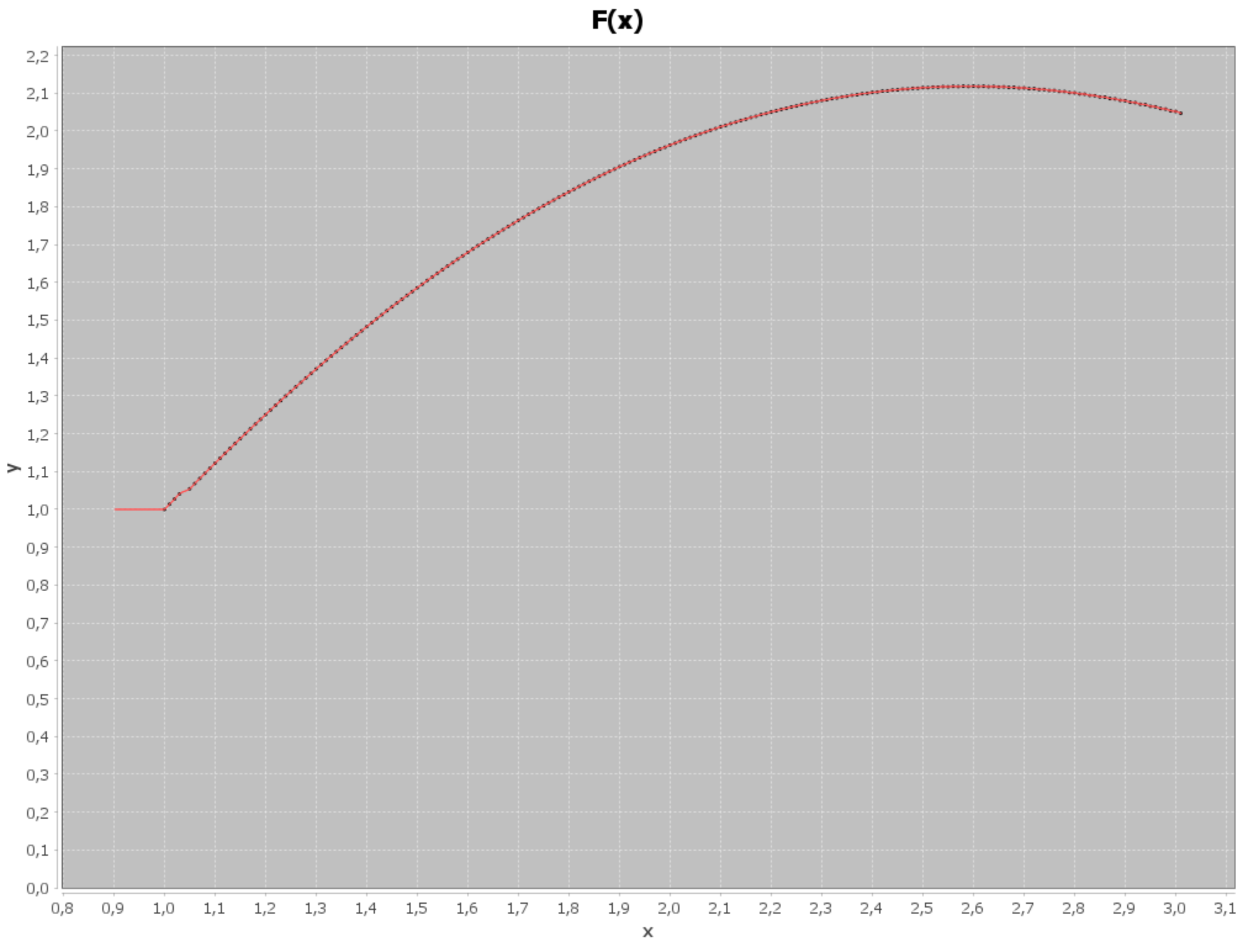
```
public List<DataPoint> applyMilneMethod(Function function, DataPoint
initPoint, double end) {
        List<DataPoint> result = new ArrayList<>();
        result.add(initPoint);
        int i = 1;
        double initX = initPoint.getX() + H;
        for (double xi = initX; xi <= end; xi += H) {
            if (i <= 3) {
                result.add(getPointByRungeKutta(function, result.get(i
- 1), xi));
            } else {
                result.add(getPointByMilne(function, result, xi, i));
            }
            i++;
        }
        return result;
    }

    private DataPoint getPointByRungeKutta(Function function,
DataPoint prev, double x) {
        double[] k = new double[4];
        k[0] = H * function.valueAt(prev.getX(), prev.getY());
        k[1] = H * function.valueAt(prev.getX() + H * 0.5, prev.getY()
+ k[0] * 0.5);
        k[2] = H * function.valueAt(prev.getX() + H * 0.5, prev.getY()
+ k[1] * 0.5);
        k[3] = H * function.valueAt(prev.getX() + H, prev.getY() +
k[2]);
        double delta = (k[0] + 2 * k[1] + 2 * k[2] + k[3]) / 6;
        return new DataPoint(x, prev.getY() + delta);
    }

   private DataPoint getPointByMilne(Function function,
List<DataPoint> points, double xi, int i) {
        double forecast = 0, correction = 0;

        double y2 = points.get(i - 2).getY();
        double y4 = points.get(i - 4).getY();
        double f1 = function.valueAt(points.get(i-1).getX(),
points.get(i-1).getY());
        double f2 = function.valueAt(points.get(i-2).getX(),
points.get(i-2).getY());
        double f3 = function.valueAt(points.get(i-3).getX(),
points.get(i-3).getY());

        correction =  y4 + 4*H/3 * (2*f3 - f2 + 2*f1);
        xi += H;
        forecast = correction;
        while (Math.abs(correction - forecast) >= ACCURACY) {
            forecast = correction;
            correction = y2 + H/3 * (f2 - 4*f1 +
2*function.valueAt(xi, forecast));
        }
        return new DataPoint(xi, correction);
    }
```

Примеры работы программы:

```
Hello, please select a function
sin(x)
sin(x) + cos(y)
log(x)
y-x^3
3*x^2*y
1
Enter x0
-4
Enter y0
0
Enter border x value
4
```



```
Hello, please select a function
sin(x)
sin(x) + cos(y)
log(x)
y-x^3
3*x^2*y
2
Enter x0
1
Enter y0
1
Enter border x value
3
```



Вывод:

Многошаговые методы обеспечивают примерно такую же точность, как и методы одношаговые. Однако в них проще оценить погрешность на шаге, в связи с чем можно выбирать большую величину шага и получать большую эффективность. Многошаговые методы также требуют использования какого либо одношагового метода для нахождения изначальных точек.