Задание на 06.03

Кириленко Андрей, ВШЭ

5 марта 2019

$\boxed{1}$

Задача Коши: $y''(x) = a^2 y(x),\ y(0) = 1,\ y'(0) = -a,\ x \in [0; T]$

Обозначим $y'(x) = z(z)$, тогда имеем систему из двух уравнений: $y'(x) = z(x),\ z'(x) = a^2 y(x)$; $z(0) = -a,\ y(0) = 1$.

$\boxed{2}$

Теперь решим полученную систему методом Эйлера. N - число интервалов, постоянный шаг $h = \frac{T}{N}$

Имеем:

$y_{i+1} = y_i + hz_i,\ z_{i+1} = z_i + a^2 h y_i$

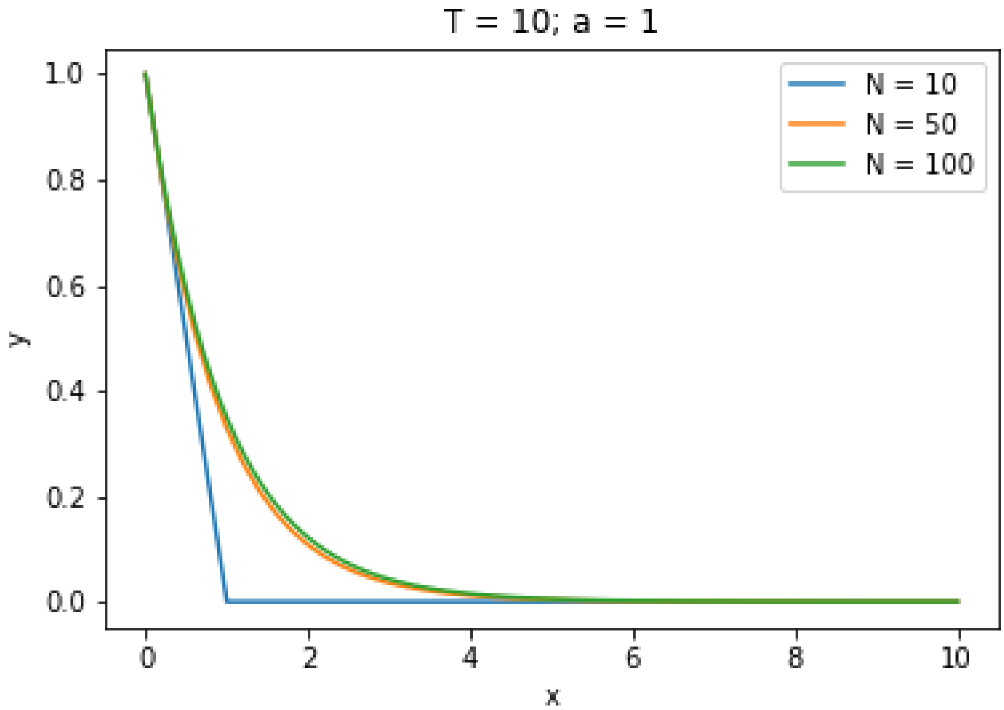
Реализуем метод Эйлера:

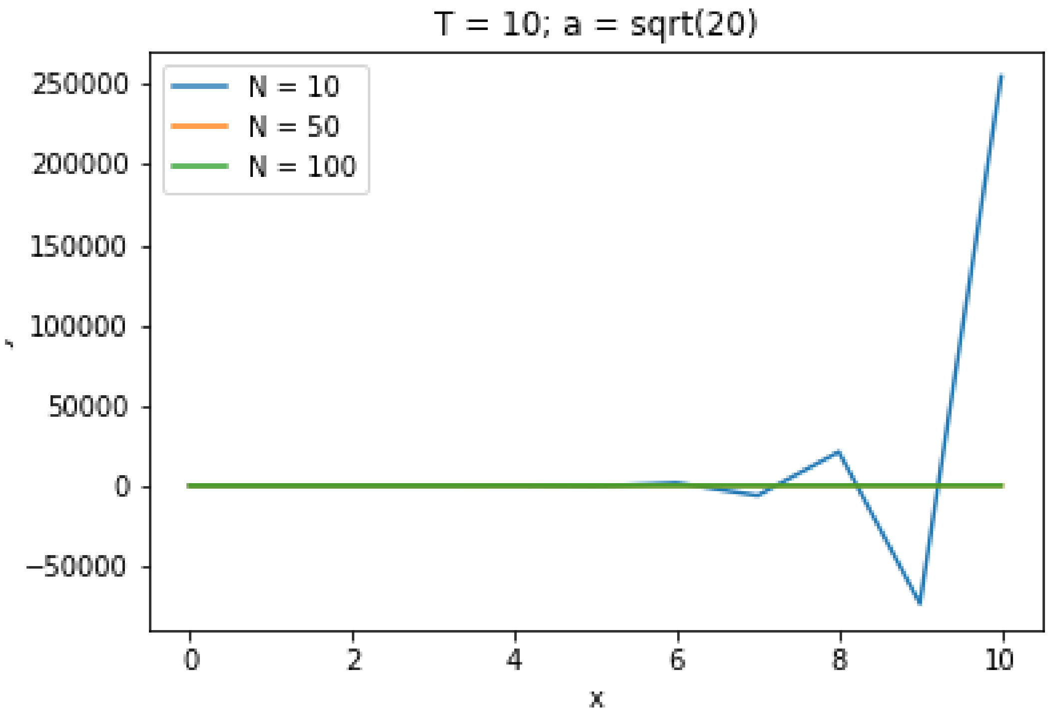
```
T=10
a=[1, math.sqrt(20)]
N=[10, 50, 100]

def euler(f0, f1, y0, y1, N):
    h = T / N
    x = 0
    y = [y0]
    z = [y1]

    for i in range(N):
        yc = y[-1] + h * f0(x, y[-1], z[-1])
        zc = z[-1] + h * f1(x, y[-1], z[-1])
        y.append(yc)
        z.append(zc)
        x += h

    return y

def task2():
    for cur_a in a:
        for cur_n in N:
            xs = [i * (T / cur_n) for i in range(cur_n + 1)]
            ys = euler(lambda x, y, z: z, lambda x, y, z: cur_a ** 2 * y, 1, -cur_a, cur_n)
            graph.plot(xs, ys, label = "N = " + str(cur_n))
            graph.ylabel("y")
            graph.xlabel("x")

        graph.title("T = 10; a = " + ("1" if cur_a == 1 else "sqrt(20)"))
        graph.legend()
        graph.savefig("hw7_t1_a" + ("1" if cur_a == 1 else "sqrt(20)") + ".png")
```

```
        graph.close()
```

Листинг 1: Метод Эйлера

А теперь построим графики:





Так как решение это убывающая экспонента, то первый график достаточно точен, кроме $N = 10$, там слишком малое число итераций и дискретизация отрезка недостаточно точная.

Аналогично, проблемы есть у $N = 10$ и на втором графике по тем же причинам. Это станет понятно из графика ошибки, при 10 она будет максимальна.

$\boxed{3}$

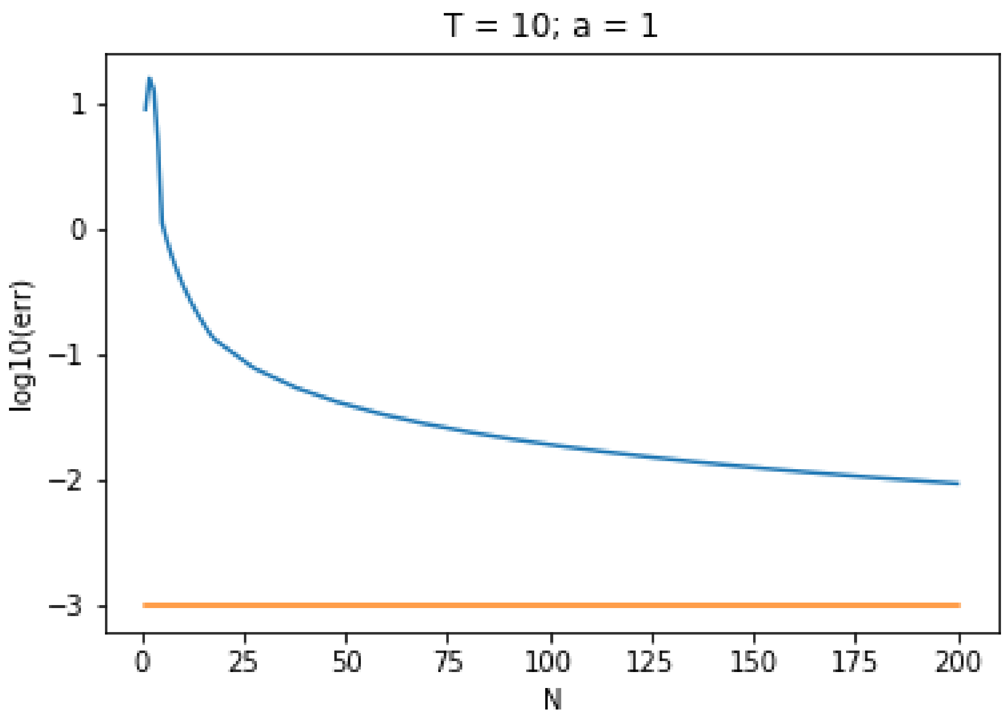
Характеристическое уравнение: $x^2 - a^2 = 0$, откуда его решения $x = a,\ x = -a$ и решение имеет вид: $y(x) = c_1 e^{ax} + c_2 e^{-ax}$

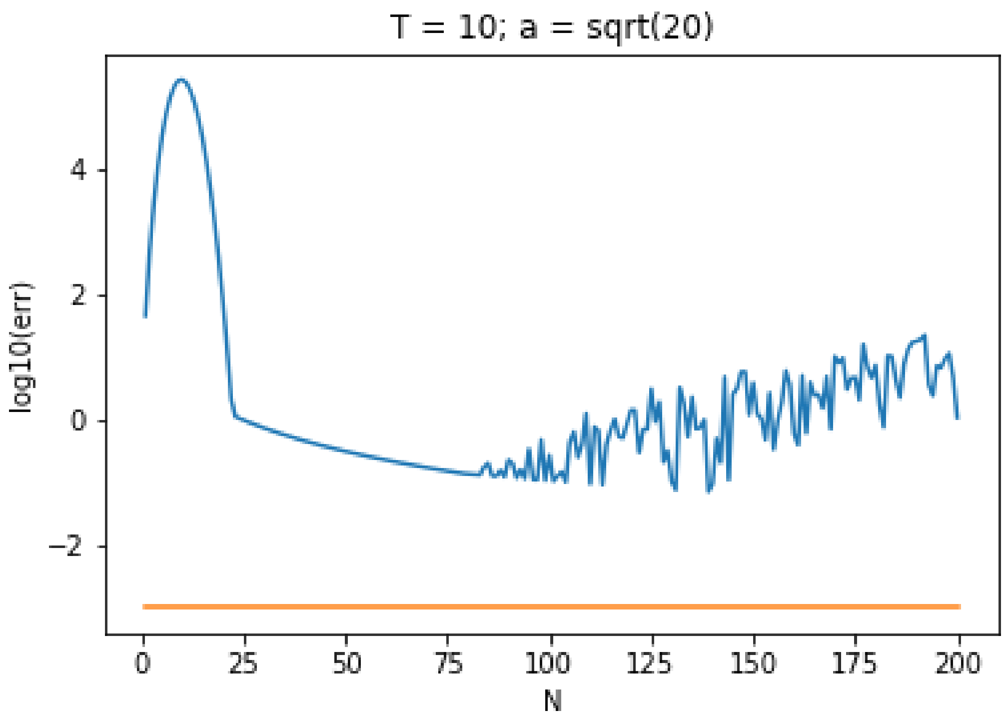
Из начальных условий имеем $c_1 + c_2 = 1, c_1 a - c_2 a = -a$, т.е. $c_1 + c_2 = 1, c_1 - c_2 = -1$, откуда $c_1 = 0,\ c_2 = 1$ и решение $y(x) = e^{-ax}$

Напишем код этой функции и реализуем метод для построение графика ошибки:

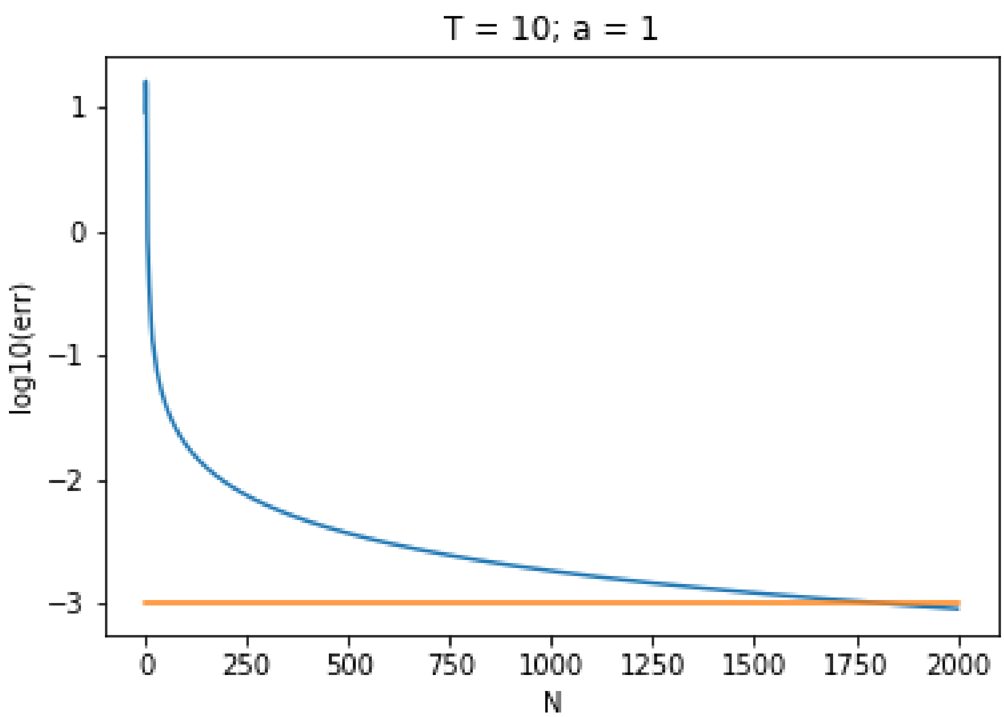
```
def task3():
    for cur_a in a:
        xs = []
        ys = []
        for n in range(1, 201):
            xs.append(n)
            h = T / n
            math_sol = [sol(cur_a)(i * h) for i in range(n + 1)]
            solution = euler(lambda x, y, z: z, lambda x, y, z: cur_a ** 2 * y, 1, -cur_a, n)
            ys.append(np.log10(max([abs(solution[i] - math_sol[i]) for i in range(len(math_sol))])))
        graph.plot(xs, ys)
        graph.ylabel("log10(err)")
        graph.xlabel("N")
        graph.plot([1, 200], [-3, -3])
        graph.title("T = 10; a = " + ("1" if cur_a == 1 else "sqrt(20)"))
        graph.savefig("hw7_t3_a" + ("1" if cur_a == 1 else "sqrt(20)") + ".png")
        graph.close()
```

Листинг 2: Построение ошибки





Сразу видно, что при $N = 10$ ошибка велика, что согласуется с предыдущей задачей. Надо лишь понять, поткуда возникает такая парабола в начале, почему сачала погрешность растет. Мне кажется, это лишь следствие недостаточной дискретизации отрезка. При $a = \sqrt{20}$ нам уже не хватает точности при больших N и желаемую отметку точности метод не достигает. Скорость убывания ошибки степенная, так как график выпрямляется. При $a = 1$ метод достигает желаемой точности, но только при $N = 1800$:



$\boxed{4}$

Реализуем метод Рунге-Кутта 2 порядка:

```
def runge(f0, f1, y0, y1, N):
    beta = 0.5
    h = T / N
    x = 0
    y = [y0]
    z = [y1]
    for i in range(N):
        yc = y[-1] + h * ( (1 - beta) * f0(x, y[-1], z[-1]) +
                              beta * f0(x + h / (2 * beta),
                                     y[-1] + h / (2 * beta) * f0(x, y[-1], z[-1]),
                                     z[-1] + h / (2 * beta) * f1(x, y[-1], z[-1])) )
        zc = z[-1] + h * ( (1 - beta) * f1(x, y[-1], z[-1]) +
                              beta * f1(x + h / (2 * beta),
                                     y[-1] + h / (2 * beta) * f0(x, y[-1], z[-1]),
                                     z[-1] + h / (2 * beta) * f1(x, y[-1], z[-1])) )
        y.append(yc)
        z.append(zc)
        x += h
    return y
```

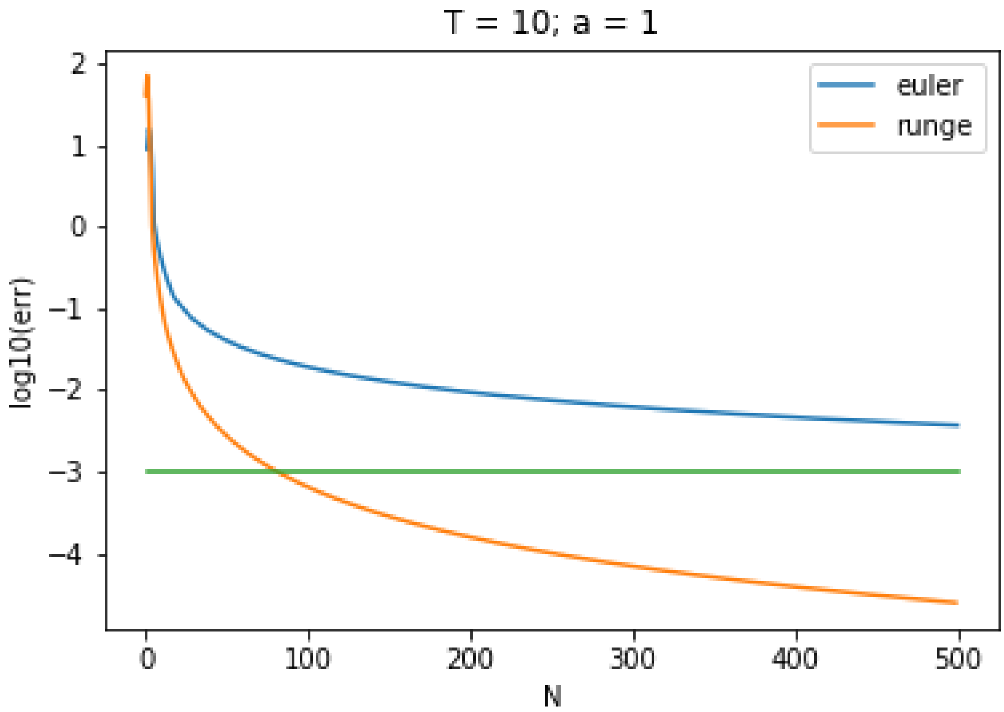
Листинг 3: Метод Рунге 2 порядка

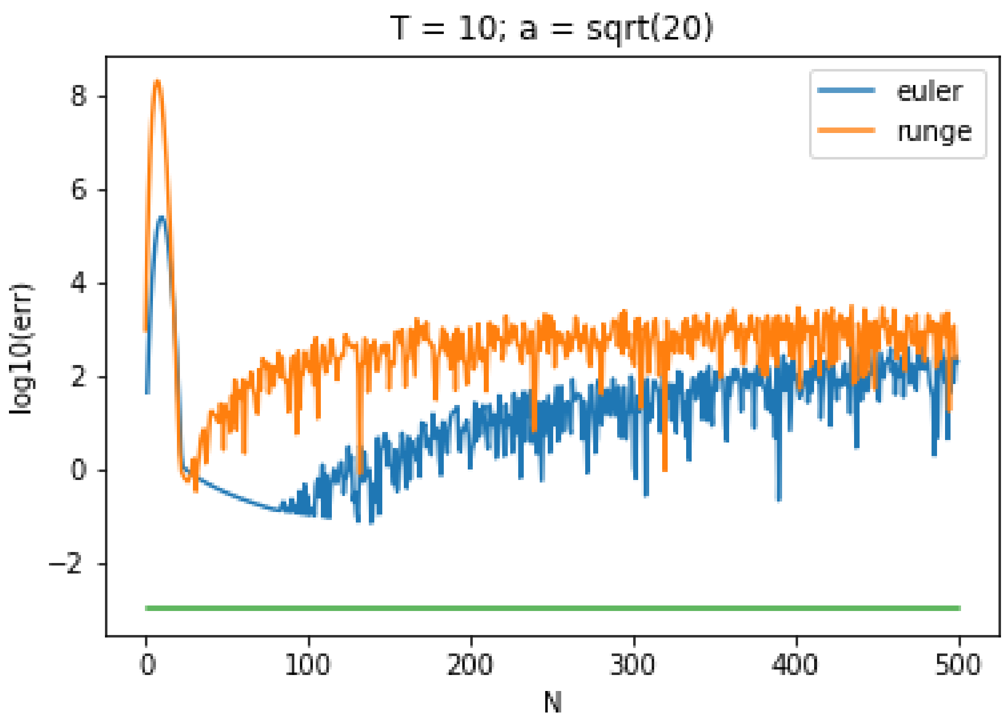
И тепень построим графики, аналогичные заданию 3:

```
def task4():
    for cur_a in a:
        xs = []
        ys1 = []
        ys2 = []
        for n in range(1, 501):
            xs.append(n)
            h = T / n
            math_sol = [sol(cur_a)(i * h) for i in range(n + 1)]
            solution1 = euler(lambda x, y, z: z, lambda x, y, z: cur_a ** 2 * y, 1, -cur_a, n)
            solution2 = runge(lambda x, y, z: z, lambda x, y, z: cur_a ** 2 * y, 1, -cur_a, n)
            ys1.append(np.log10(max([abs(solution1[i] - math_sol[i]) for i in range(len(math_sol))])))
            ys2.append(np.log10(max([abs(solution2[i] - math_sol[i]) for i in range(len(math_sol))])))
        graph.plot(xs, ys1, ys2)
        graph.ylabel("log10(err)")
        graph.xlabel("N")
        graph.legend(("euler", "runge"))
        graph.plot([1, 500], [-3, -3])
```

```
        graph.title("T = 10; a = " + ("1" if cur_a == 1 else "sqrt(20)"))
        graph.savefig("hw7_t4_a" + ("1" if cur_a == 1 else "sqrt(20)") + ".png")
        graph.close()
```

Листинг 4: Построение ошибки





Опять же, для корня возникают проблемы с точностью, причем у Рунге они проявляются даже раньше. Для $a = 1$ же, метод Рунге сходится быстрее и достигает желаемой точности уже при $N = 100$. Проблемы возникают именно с точностью, поскольку постоянно наращиваем неудобные для компьютера числа быстрыми темпами. Я это продемонстрирую, но после того как напишу метод Рунге 4 порядка:

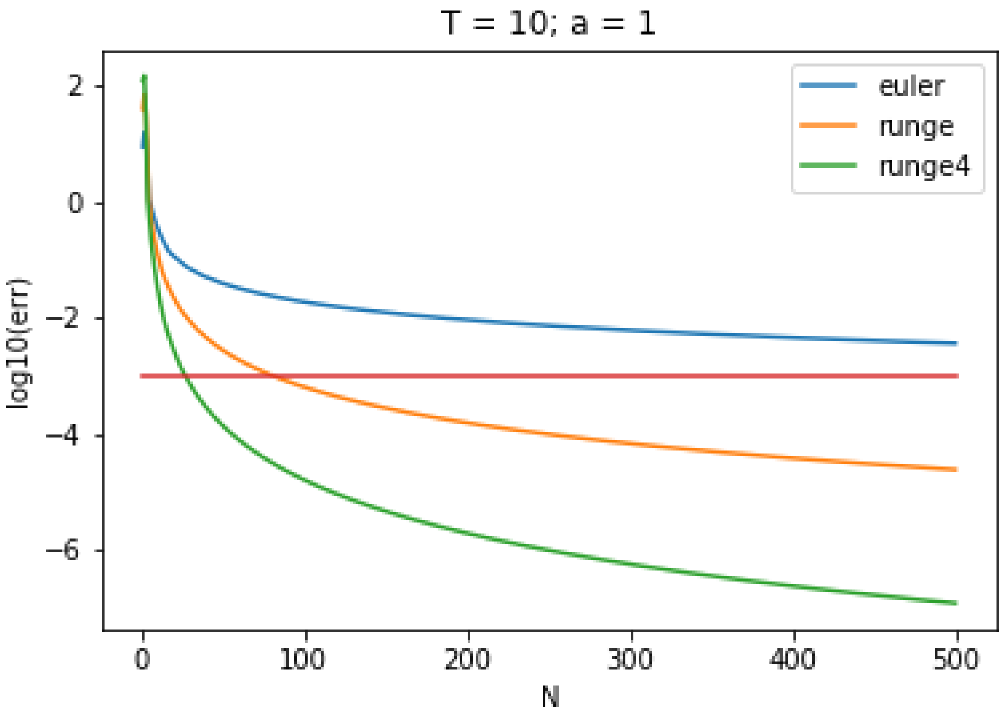
```
def runge4(f0, f1, y0, y1, N):
    h = T / N
    x = 0
    y = [y0]
    z = [y1]
    for i in range(N):
        yk1 = f0(x, y[-1], z[-1])
        zk1 = f1(x, y[-1], z[-1])
        yk2 = f0(x + h / 2, y[-1] + h * yk1 / 2, z[-1] + h * zk1 / 2)
        zk2 = f1(x + h / 2, y[-1] + h * yk1 / 2, z[-1] + h * zk1 / 2)
        yk3 = f0(x + h / 2, y[-1] + h * yk2 / 2, z[-1] + h * zk2 / 2)
        zk3 = f1(x + h / 2, y[-1] + h * yk2 / 2, z[-1] + h * zk2 / 2)
        yk4 = f0(x + h, y[-1] + h * yk2, z[-1] + h * zk2)
        zk4 = f1(x + h, y[-1] + h * yk2, z[-1] + h * zk2)
        yc = y[-1] + (h / 6.0) * (yk1 + 2 * yk2 + 2 * yk3 + yk4)

        zc = z[-1] + (h / 6.0) * (zk1 + 2 * zk2 + 2 * zk3 + zk4)
        y.append(yc)
        z.append(zc)
        x += h
    return y
```

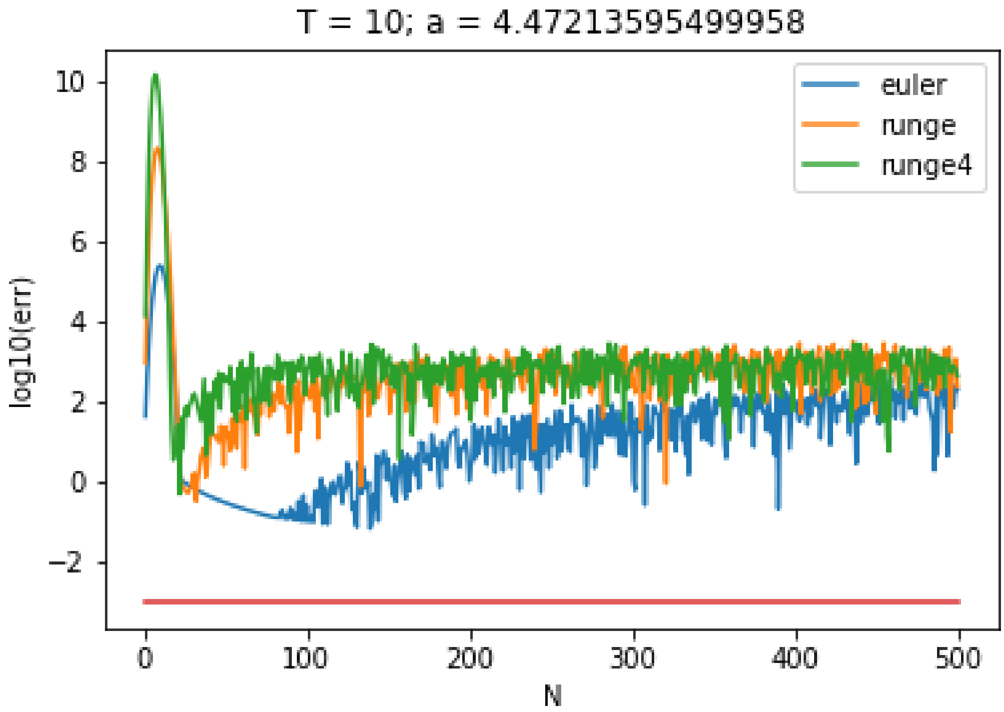
Листинг 5: Метод Рунге 4 порядка

```
def task4b():
    for cur_a in a:
        xs = []
        ys1 = []
        ys2 = []
        ys3 = []
        for n in range(1, 501):
            xs.append(n)
            h = T / n
            math_sol = [sol(cur_a)(i * h) for i in range(n + 1)]
            solution1 = euler(lambda x, y, z: z, lambda x, y, z: cur_a ** 2 * y,
 1, -cur_a, n)
            solution2 = runge(lambda x, y, z: z, lambda x, y, z: cur_a ** 2 * y,
 1, -cur_a, n)
            solution3 = runge4(lambda x, y, z: z, lambda x, y, z: cur_a ** 2 * y
, 1, -cur_a, n)
            ys1.append(np.log10(max([abs(solution1[i] - math_sol[i]) for i in
range(len(math_sol))])))
            ys2.append(np.log10(max([abs(solution2[i] - math_sol[i]) for i in
range(len(math_sol))])))
            ys3.append(np.log10(max([abs(solution3[i] - math_sol[i]) for i in
range(len(math_sol))])))
        graph.plot(xs, ys1)
        graph.plot(xs, ys2)
        graph.plot(xs, ys3)
```

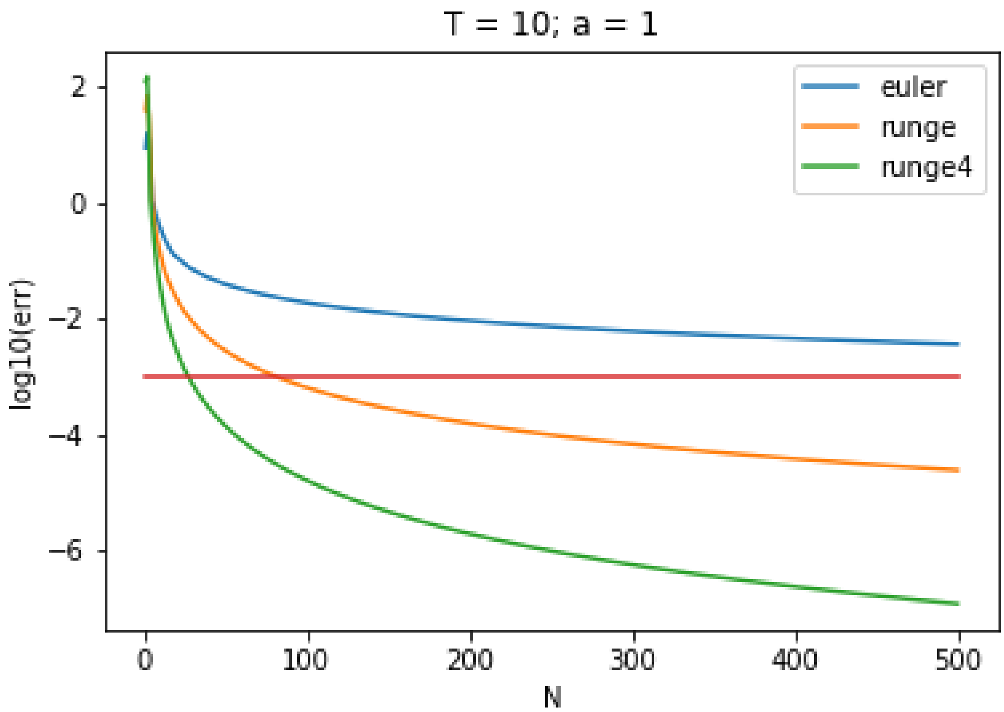
```
        graph.ylabel("log10(err)")
        graph.xlabel("N")
        graph.legend(("euler", "runge", "runge4"))
        graph.plot([1, 500], [-3, -3])
        graph.title("T = 10; a = " + str(cur_a))
        #graph.show()
        graph.savefig("hw7_t4b_a" + ("1" if cur_a == 1 else "sqrt(20)") + ".png"
    )
        graph.close()
```

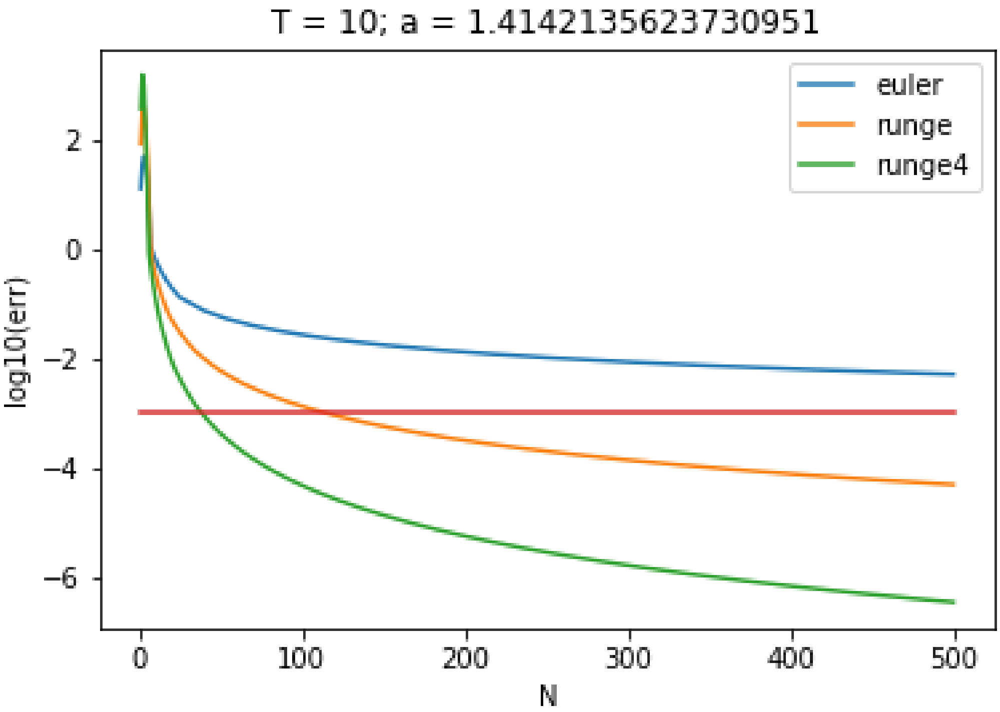
Листинг 6: Построение ошибки

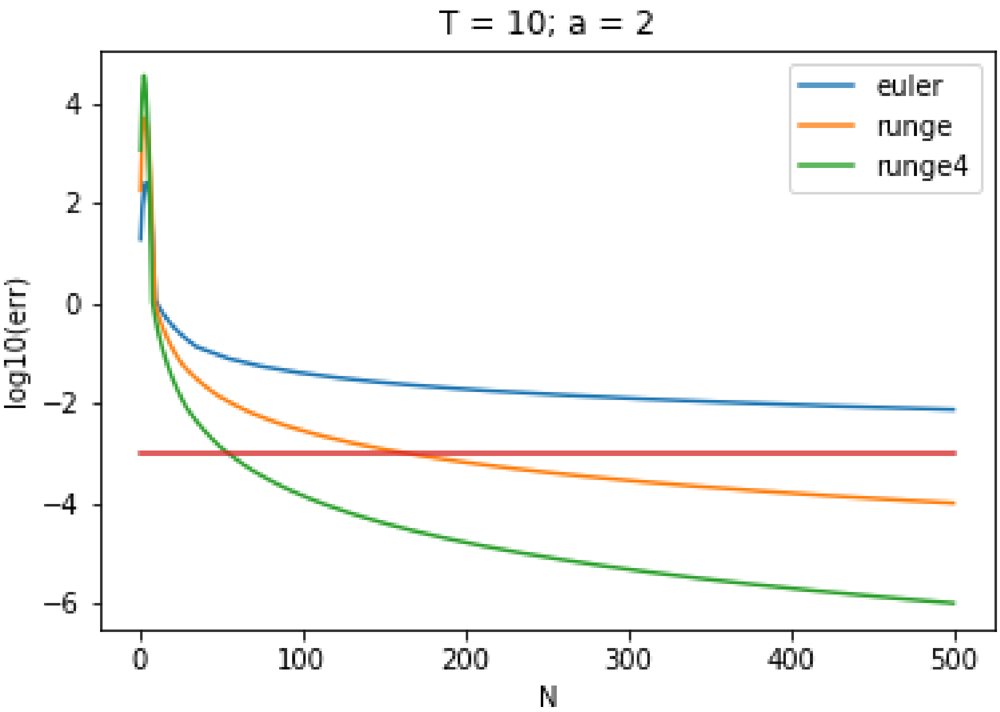


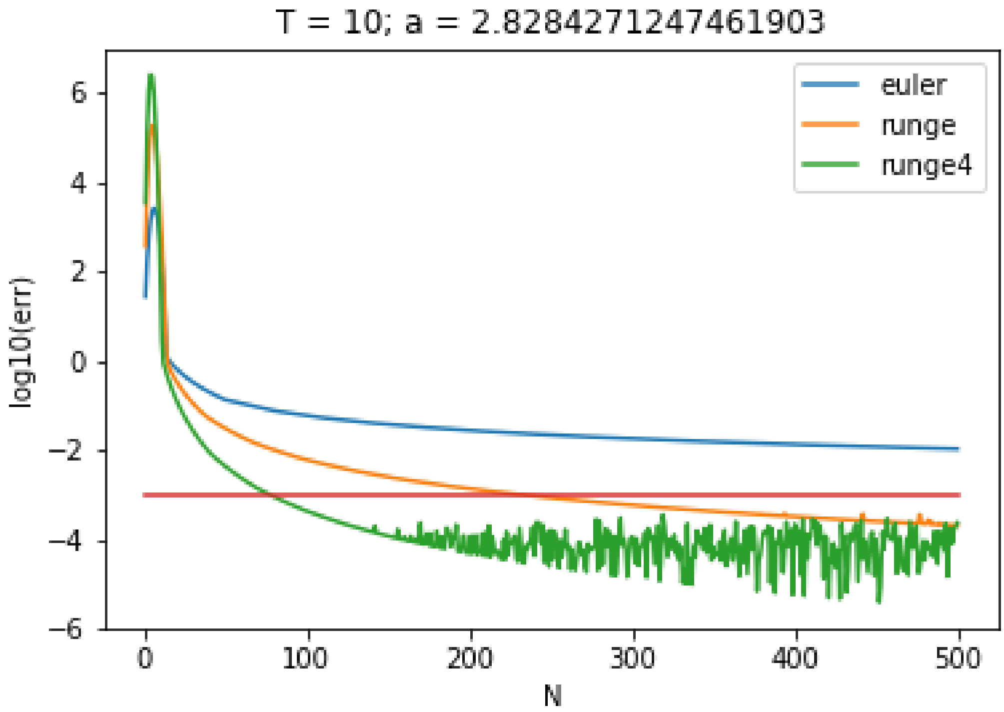


Сразу видно, что метод сходится еще быстрее, но одновременно еще больше страдает от точности. Построим серию графиков:

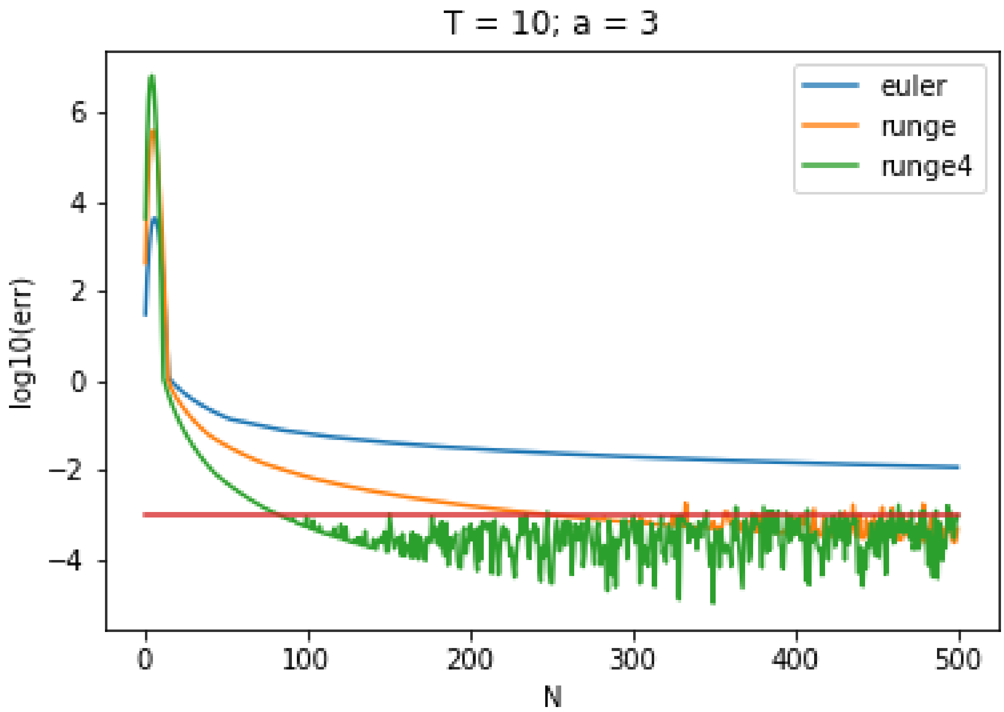



T = 10; a = 1.4142135623730951
euler
runge
runge4
log10(err)
N
2
0
−2
−4
−6
0
100
200
300
400
500

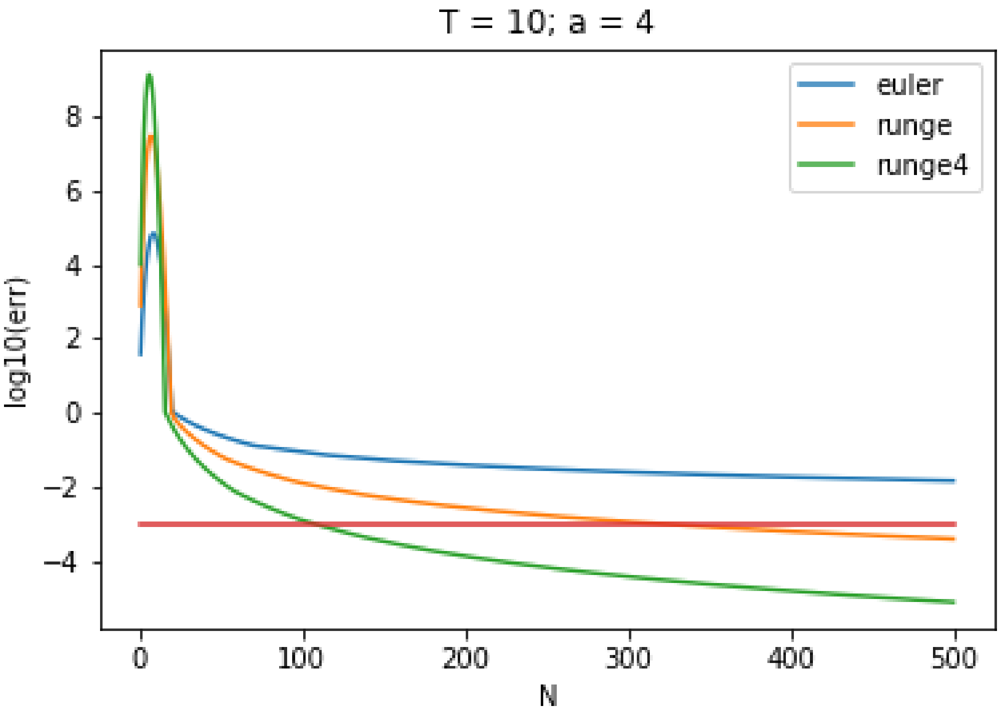

T = 10; a = 2
euler
runge
runge4
log10(err)
N
4
2
0
−2
−4
−6
0
100
200
300
400
500

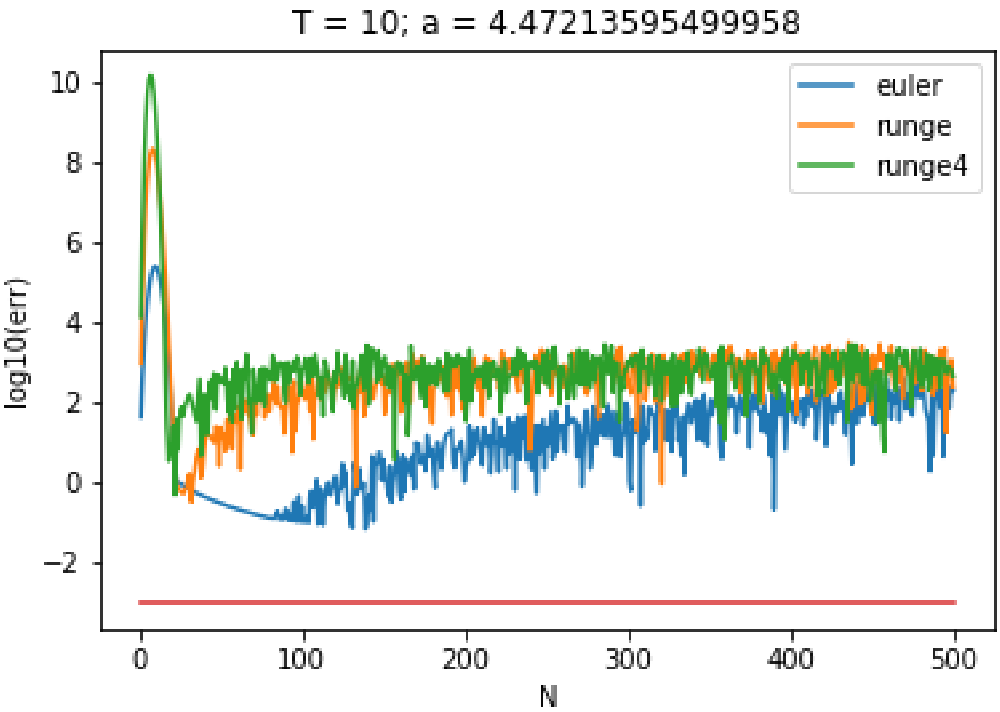


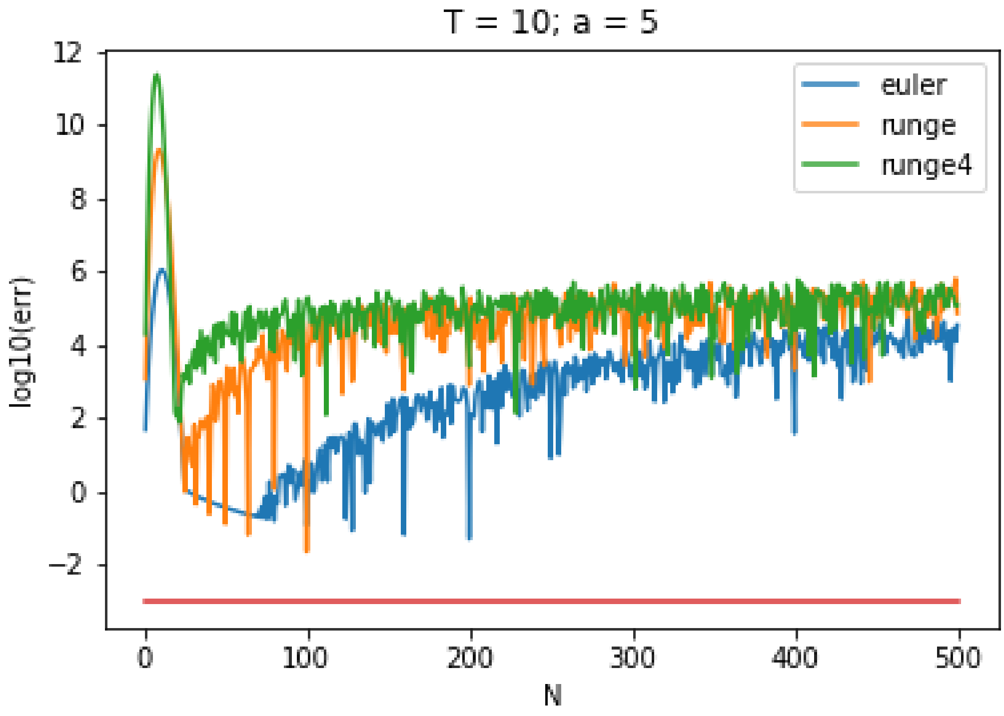
T = 10; a = 2.8284271247461903
euler
runge
runge4
log10(err)
N

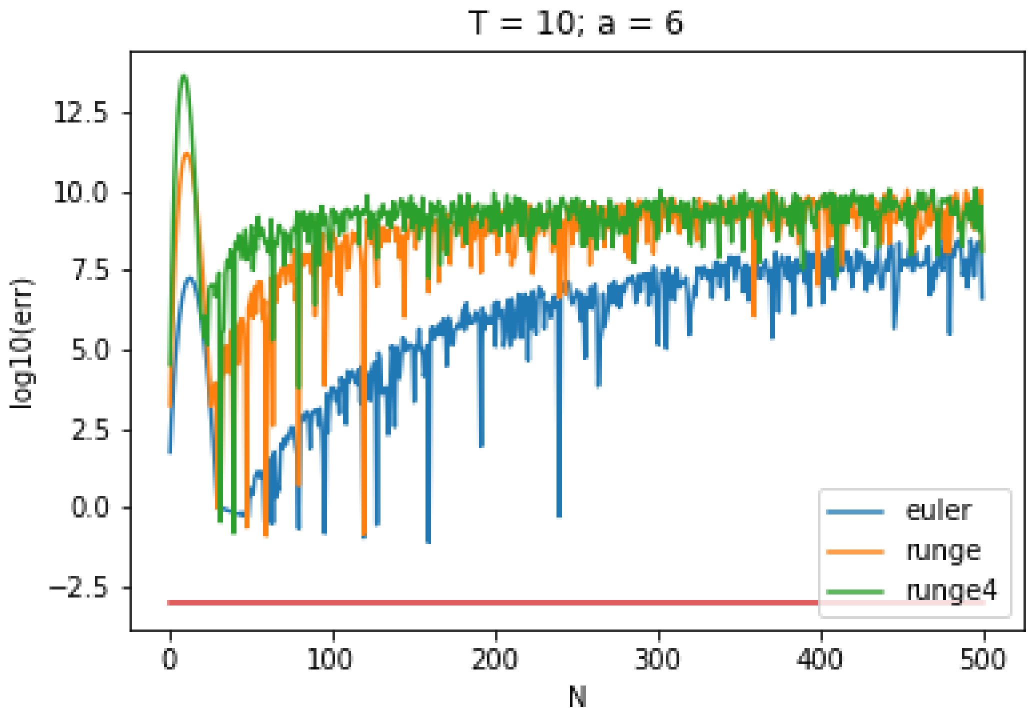


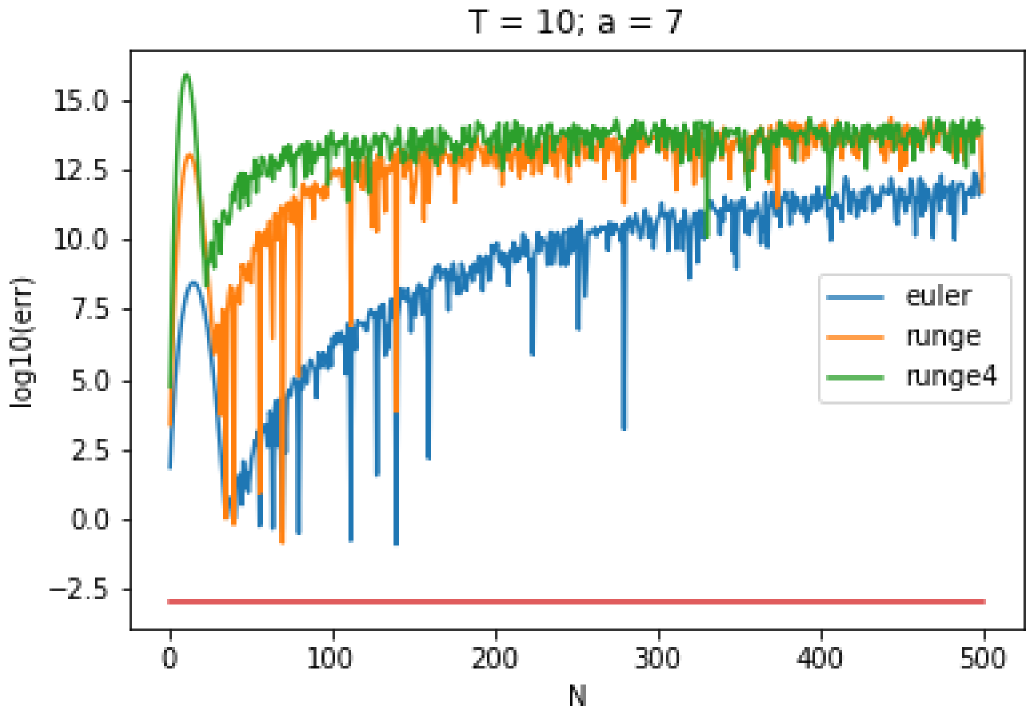
T = 10; a = 3
euler
runge
runge4
log10(err)
N

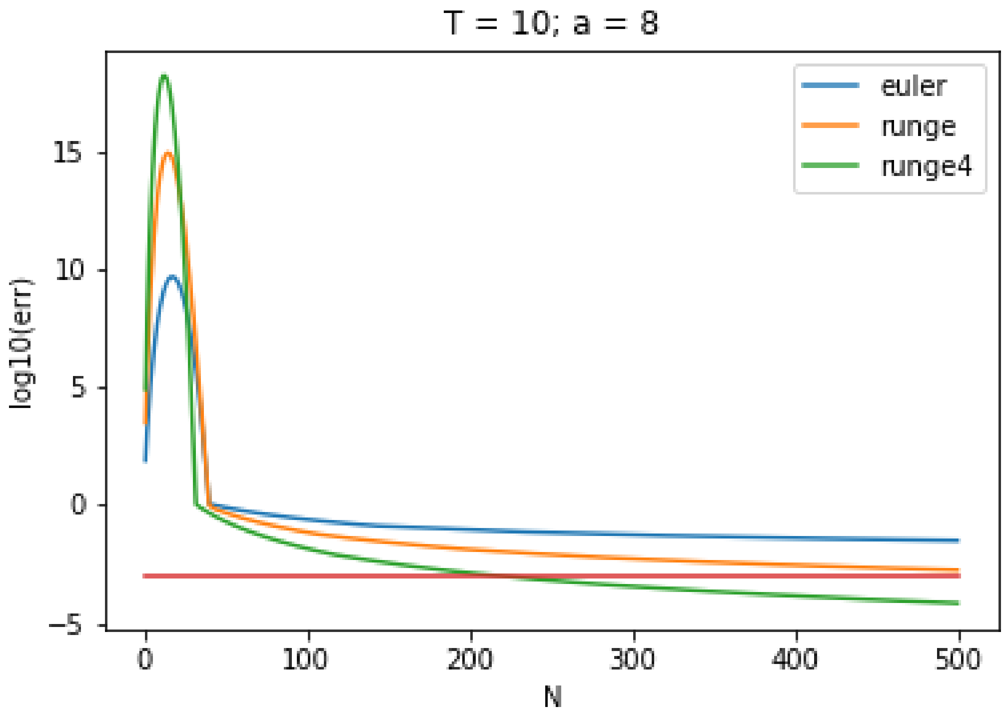

T = 10; a = 4
euler
runge
runge4
log10(err)
N
8
6
4
2
0
−2
−4
0
100
200
300
400
500

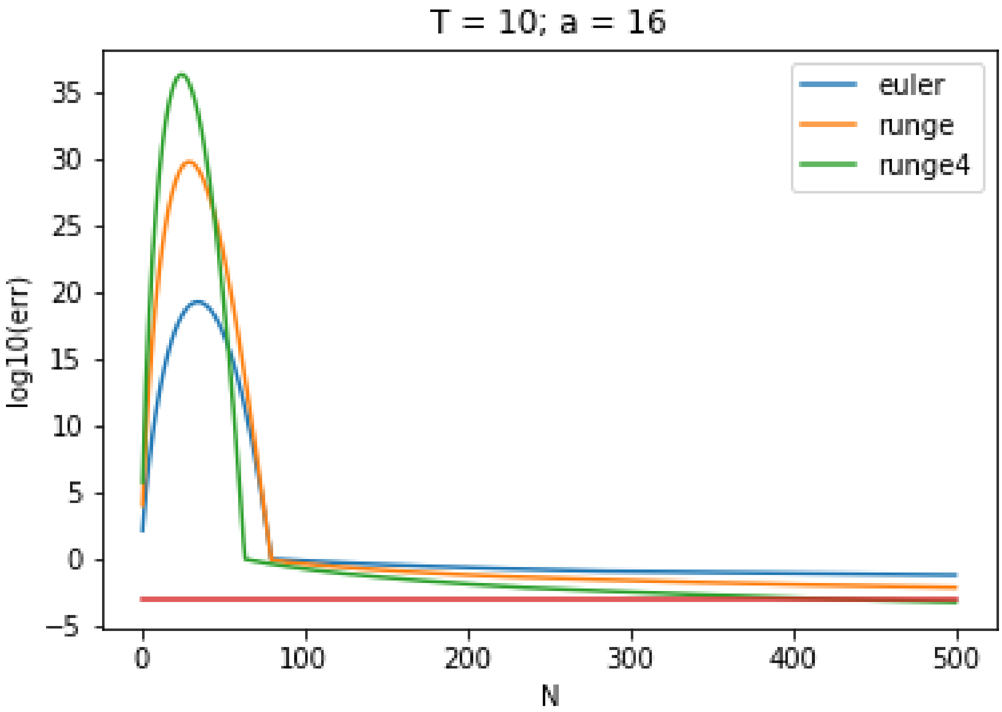

T = 10; a = 4.47213595499958
euler
runge
runge4
log10(err)
N
10
8
6
4
2
0
−2
0
100
200
300
400
500

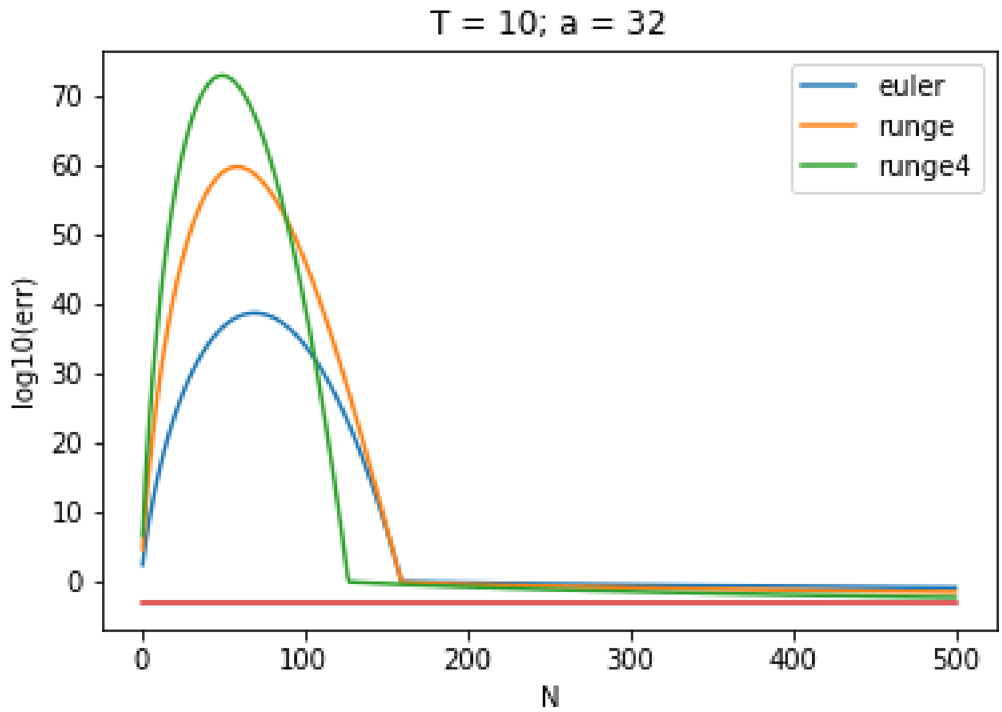

T = 10; a = 5
log10(err)
N
euler
runge
runge4


T = 10; a = 6
log10(err)
N
euler
runge
runge4


T = 10; a = 7
log10(err)
N
euler
runge
runge4
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
0
100
200
300
400
500


T = 10; a = 8
log10(err)
N
euler
runge
runge4
15
10
5
0
−5
0
100
200
300
400
500





Как итог: для удобных чисел (степени двойки) и для малых чисел, Методы рунге отлично себя проявляют, но для плохили чисел точность при постоянном шаге очень быстро теряется.