Задание на 15.02

Кириленко Андрей, ВШЭ

14 февраля 2019

$\boxed{1}$

Для поиска корней будем использовать метод Ньютона.
Вычисление многочленов Лежандра и их производной: (можно было просто использоать метод $deriv()$ для производной)

```
def p(n, x):
    return ss.legendre(n)(x)

def dp(n, x):
    return n / (1 - x*x) * (p(n-1, x) - x*p(n, x))
```

Листинг 1: Формулы

Далее, будем считать корни. Чтобы посчитать корни для итерации n, нужно знать корни на итерации $n-1$, чтобы воспользоваться перемежаемостью.
Стартовать на каждом отрезке будем из его середины.

```
ITERS = 300

def calc_roots(prev_roots):
    n = len(prev_roots) + 1
    pnts = []
    pnts.append(-1)
    pnts.extend(prev_roots)
    pnts.append(1)

    res = []

    for i in range(1, len(pnts)):
        prev_x = pnts[i-1]
        cur_x = pnts[i]
        x0=(prev_x+cur_x)/2
        for j in range(ITERS):
            x0 = x0 - p(n, x0)/dp(n, x0)
        res.append(x0)
    return res

def get_roots(n):
    r0 = []
    for i in range(n):
        r0 = calc_roots(r0)
    return r0
```

Листинг 2: Метод Ньютона - корни

Далее, зная корни, требуется посчитать веса.
Для этого была формула: $\int_a^b \prod_{k\neq i} \frac{x-x_k}{x_i-x_k}\rho(x)dx$, $\rho(x) = 1$
Собственно, считаем, методом Симпсона:

```
def gen_f(i, N, r):
    def f(x):
        ans = 1
        for k in range(N):
            if k != i:
                ans *= ((x - r[k])/(r[i]-r[k]))
        return ans
    return f

def calc_w(roots):
    n = len(roots)
    res = []
    for i in range(n):
        res.append(simpson(gen_f(i, n, roots), -1, 1, ITERS))
    return res
```

Листинг 3: Подсчет весов

Зная веса, несложно посчитать интеграл по Квадратурной формуле, также стоит учесть, что надо отобразить отрезок в $[-1;\ 1]$:

```
def func(x):
    return 1 / (9 * x ** 2 + 1)

def calc_integral(f, a, b, n):
    roots = get_roots(n)
    wi = calc_w(roots)
    summ = 0
    for i in range(n):
        summ += wi[i] * f((b - a) / 2 * roots[i] + (a + b) / 2)

    return summ * (b - a) / 2
```

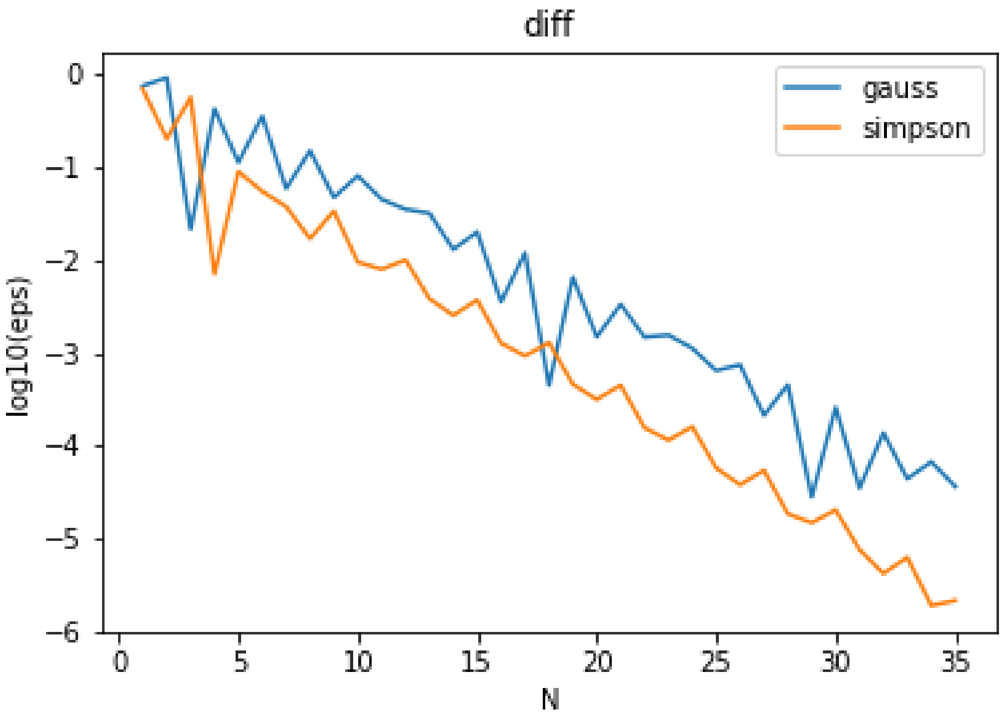
Листинг 4: Вычисление интеграла

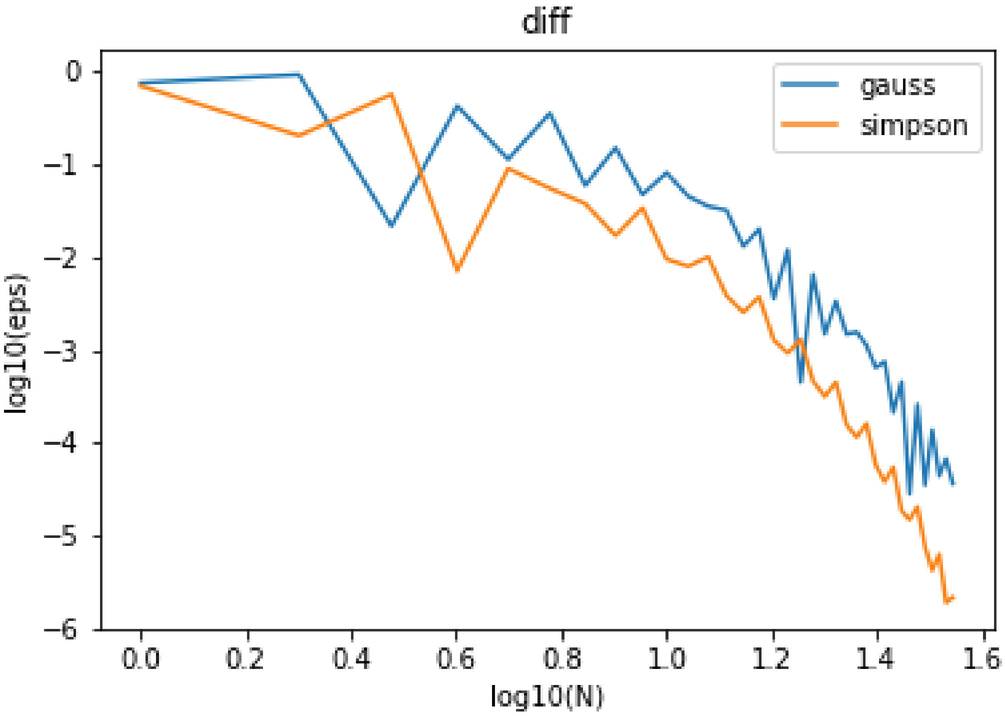
Посмотрим на графики ошибок и сравним их с уже реализованным методом Симпсона:

```
def plot_error():
    real_res = 0.9177579784724424
    y1 = []
    y2 = []
    x = []
    for n in range(1, 56):
        print(n)
        i1 = calc_integral(func, -1, 5, n)
        i2 = simpson(func, -1, 5, n)
```

```python
        err1 = np.log10(abs(i1-real_res))
        err2 = np.log10(abs(i2-real_res))
        x.append(np.log10(n))
        y1.append(err1)
        y2.append(err2)
    graph.plot(x, y1, label='gauss')
    graph.plot(x, y2, label='simpson')
    graph.xlabel('log10(N)')
    graph.ylabel('log10(eps)')
    graph.title('diff')
    graph.legend()
    graph.savefig("t1logx.png")
    graph.close()
```

Листинг 5: Код графиков





Можно заметить, что аналогично Симпсону, ошибка убывает экспоненциально, однако несколько хуже по точности.

Точность этого метода скорее соответствует методу трапеций, если смотреть график.

Также стоит заметить, что для больших значений N (> 35) вычисления долгие и с переполнениями, поэтому объективность рассуждений для больших значений весьма сомнительна.