Задание на 06.02

Кириленко Андрей, ВШЭ

5 февраля 2019

$\boxed{1}$

Выведем формулу второго порядка точности для первой производной. Обозначим $x_1 = x_0 - \frac{h}{a}$, $x_2 = x_0$, $x_3 = x_0 + ha$. Так как нужен второй порядок, оставляем два первых слагаемых в формуле производной: $p'(x) = [x_1, x_2]f + [x_1, x_2, x_3]f * (x - x_1 + x - x_2)$.

Подставляем разделенные разности, получаем:

$f'(x_0) \sim p'(x_2) = \frac{f(x_2)-f(x_1)}{x_2-x_1} + \frac{\frac{f(x_3)-f(x_1)}{x_3-x_1} - \frac{f(x_2)-f(x_1)}{x_2-x_1}}{x_3-x_2} * (x_2 - x_1) = \frac{y_2-y_1}{h/a} + \frac{\frac{y_3-y_1}{ah+h/a} - \frac{y_2-y_1}{h/a}}{ah} * \frac{h}{a} = \frac{1}{h}((y_2 - y_1)(a - \frac{1}{a}) + \frac{y_3-y_1}{a^3+a})$.

Теперь можно построить графики для синуса.

Пусть $\alpha = 1, 1.2, 1.4, 1.6, 1.8, 2$, $f(x) = sin(x)$, $f'(x) = cos(x)$, $h \in [10^{-8}, 1]$

Напишем код построения графиков:

Расчеты:

```
def f(x):
    return np.sin(x)

def df(x):
    return np.cos(x)

def p(a, h, x0):
    x1 = x0 - h / a
    y1 = f(x1)
    x2 = x0
    y2 = f(x2)
    x3 = x0 + h * a
    y3 = f(x3)
    return 1 / h * ((y2 - y1) * (a - 1 / a) + (y3 - y1) / (a ** 3 + a))
```

Листинг 1: Вычисления

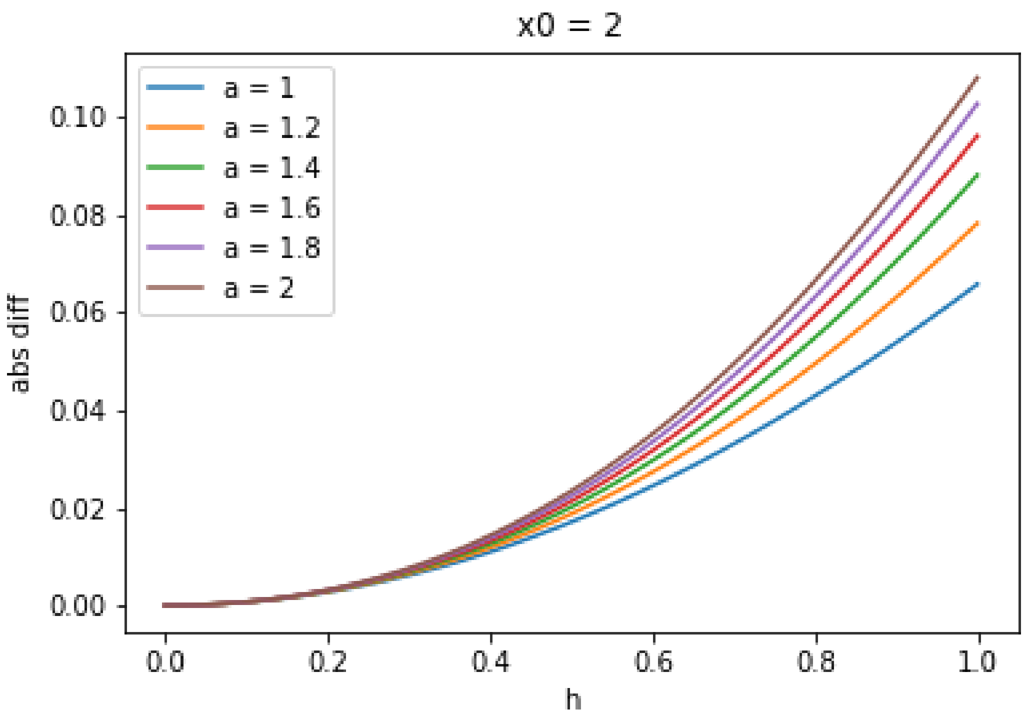
Далее построим графики с помощью следующего кода:

```
def plot_error(alph, x0):
    x = []
    y = []
    step = 10 ** (-3)
    h = 10 ** (-8)
    b = 1
    ans = df(x0)
    while h <= b:
        x.append(h)
        y.append(abs(p(alph, h, x0) - ans))
        h += step
    graph.title("x0 = " + str(x0))
    graph.plot(x, y)
```

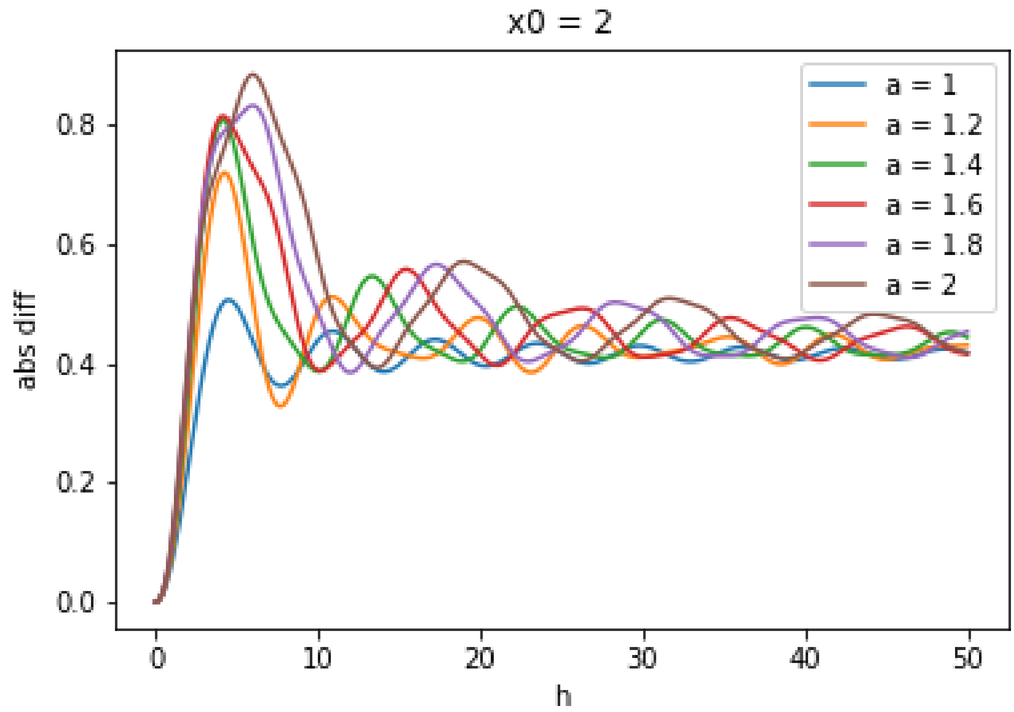
```
    graph.ylabel("abs diff")
    graph.xlabel("h")

def task1():
    x0 = 2
    arr = [1, 1.2, 1.4, 1.6, 1.8, 2]

    for a in arr:
        plot_error(a, x0)
```

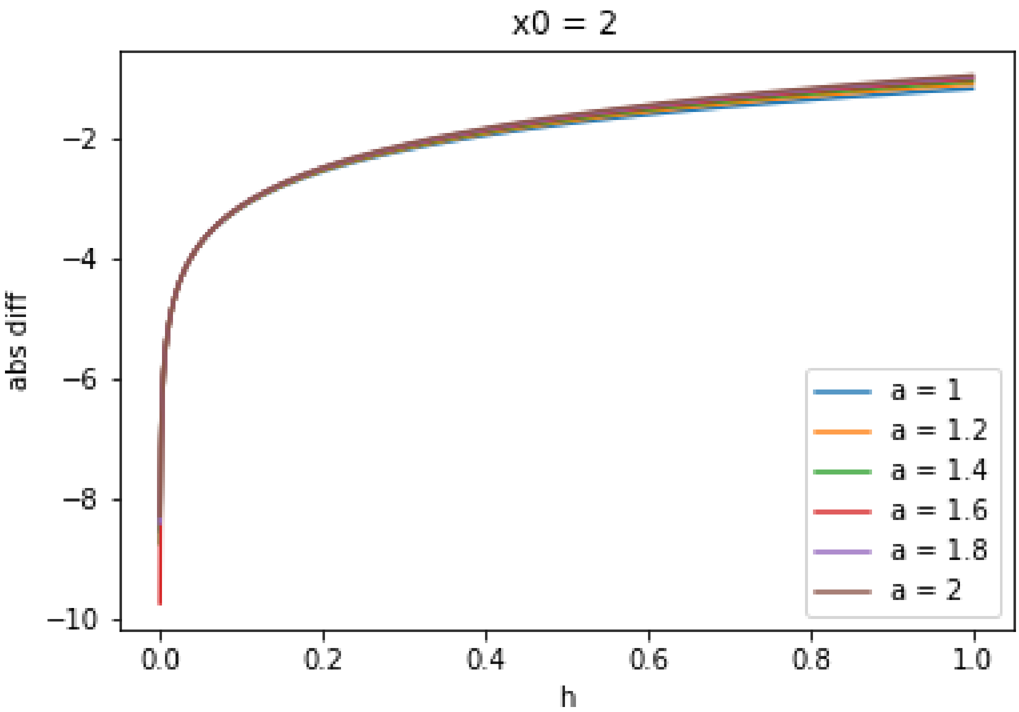
Листинг 2: Построение графиков

Получим следующее:



Также попытаемся построить логарифм ошибки, а также попробуем увеличить h:





Обсудим полученные результаты. Во-первых, в оценку погрешности по модулю подставим $N = 2, n = 1$, получим оценку $C * \frac{||f^{(3)}||_C}{2} * max_i|x - x_i|^2$. Норма синуса это единица, максимум достигается в точке $x_0 + \alpha h$, так как $\alpha \geq 1$. Итого оценка из теории это $O(\alpha^2 h^2)$.
И действительно, если посмотреть на первый график, действительно видим параболу. Кроме того, ошибка прямо пропорционалана α, как и в оценке.
График логарифма в данном случае не интересен, степенной рост дает логарифмический график, и они почти совпадают.
Наконец, если посмотреть на расширенный график, можно отметить две вещи: погреш-

ность перестает быть параболой, и становится периодической, я это связываю с тем, что так как синус периодичен, то мы попадаем в “похожие” точки, но сдвинутые на период. Кроме того, есть тенденция стремления погрешности к истинному значению модуля производной, так как приблизительная производная стремится к нулю из-за множителя $\frac{1}{h}$, а остальное ограничено, и значение $|f' - approx| -> |f' - 0| -> |f'|$.

$\boxed{\text{2a}}$

Посчитаем точное значение интеграла:

$\int_{-1}^{5} \frac{1}{9x^2+1} = \frac{1}{3}arctg(3x)|_{-1}^{5} \sim 0.9177579784724423$

Теперь реализуем методы трапеций и Симпсона:

```
def func(x):
    return 1.0 / (9 * (x ** 2) + 1)

def func_int_indefinite(x):
    return np.arctan(3 * x) / 3

def func_int_definite(a, b):
    return func_int_indefinite(b) - func_int_indefinite(a)

def trapezioid(f, a, b, M):
    summ = (f(a) + f(b)) / 2
    h = (b - a) / M
    x = a + h
    for i in range(1, M):
        summ += f(x)
        x += h
    return summ * h

def simpson(f, a, b, M):
    h = (b - a) / M
    summ = f(a) + f(b) + 4 * f(a + h / 2)
    x = a + h
    for i in range(1, M):
        summ += 2 * f(x) + 4 * f(x + h / 2)
        x += h
    return summ * (h / 6)
```

Листинг 3: Методы трапеций и Симпсона

Сделаем вычисление всеми способами:

```
    print("Math answer:")
    print(func_int_definite(-1, 5))
    print("Trapezioid answer:")
    print(trapezioid(func, -1, 5, 1000))
    print("Simpson answer:")
    print(simpson(func, -1, 5, 1000))
    print("|math - trapezioid|:")
    print(abs(trapezioid(func, -1, 5, 1000) - func_int_definite(-1, 5)))
```

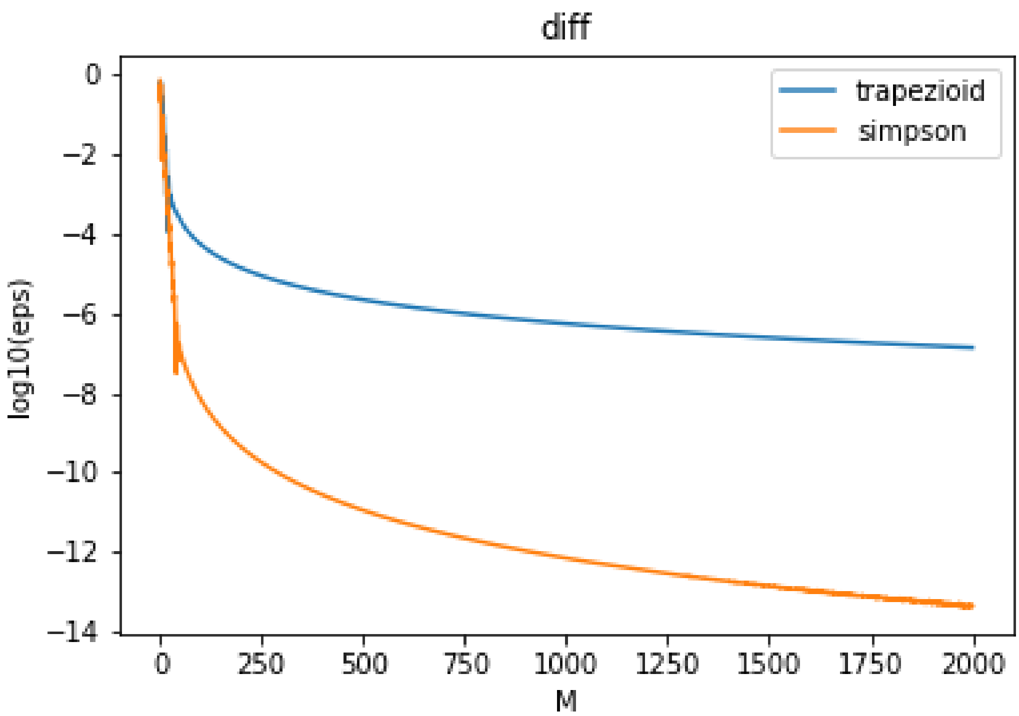
```
    print("|math - simpson|:")
    print(abs(simpson(func, -1, 5, 1000) - func_int_definite(-1, 5)))
```

Листинг 4: Вычисление интеграла

```
Math answer:
0.9177579784724423
Trapezioid answer:
0.9177574331890018
Simpson answer:
0.9177579784717418
|math - trapezioid|:
5.452834405117457e-07
|math - simpson|:
7.005507285384738e-13
```

Листинг 5: Результаты

Как легко видеть, Симпсон получился на порядки точнее. Займемся графиками:



Шкала логарифмическая, при увеличении правой границы начинаются деления на 0 в логарифме, поэтому строить для больших значений, чем 2000, нецелесобразно из-за машинной точности.

Доказанная оценка была для трапций $err \leq O(\frac{1}{M^2})$, для Симпсона $err \leq O(\frac{1}{M^4})$, значит для логарифмов:

трапеции: $log_{10}(err) \leq C_1 - 2 * \log_{10}(M))$, Симпсон: $log_{10}(err) \leq C_2 - 4 * \log_{10}(M))$. Эти оценки полностью соответствуют графику, и Симпсон действительно точнее.

$\boxed{\text{2b}}$

$\epsilon = 10^{-6}$

$10^{-3} = H_2 = \frac{H_1}{2}$

$H \leq \sqrt{\frac{\epsilon}{C}}$, $C = \frac{1}{3H_2^2}(S_{H_2} - S_{H_1})$

Напишем код, считающий это:

```
def calc_h_runge(eps, a, b):
    h2 = 10 ** (-3)
    h1 = 2 * h2
    sh2 = trapezioid(func, a, b, int((b - a) / h2))
    sh1 = trapezioid(func, a, b, int((b - a) / h1))
    c = abs(1 / (3 * h2 ** 2) * (sh2 - sh1))
    return np.sqrt(eps / c)

def calc_h(eps, a, b):
    start = 1
    finish = 2000
    x = []
    yt = []
    ans = func_int_definite(a, b)
    while start <= finish:
        x.append(start)
        yt.append(abs(ans - trapezioid(func, a, b, start)))
        start += 1
    for i in range(1, len(yt)):
        if yt[i] < eps:
            return (b - a) / x[i]
```

Листинг 6: Вычисление размера интервала

```
    print("Runge:")
    print(calc_h_runge(10 ** (-6), -1, 5))
    print("Real:")
    print(calc_h(10 ** (-6), -1, 5))
    print("Diff:")
    print(abs(calc_h_runge(10 ** (-6), -1, 5) - calc_h(10 ** (-6), -1, 5)))

Runge:
0.008125294581500147
Real:
0.008119079837618403
Diff:
6.214743881743576e-06
```

Листинг 7: Результаты

Комментарий: во-первых, реальный результат чуть-чуть меньше, что уже хорошо, Рунге дает верхнюю оценку. Во-вторых, оценка в данном случае оказалась точной, и она очень близка к идеальной.

$\boxed{2c}$

Напишем код для вычисления весов и первого отрицательного веса:

```
def gen_f(i, N):
    def f(q):
```

```python
        ans = 1
        for k in range(1, N + 1):
            if k != i:
                ans *= q - (k - 1)
        return ans
    return f

def fac(n):
    return math.factorial(n)

def calc_weights(a, b, N):
    h = (b - a) / (N - 1)
    arr = []
    for i in range(1, N + 1):
        li = simpson(gen_f(i, N), 0, N - 1, 1000)
        li *= (-1) ** (N - i) * h / fac(i - 1) / fac(N - i)
        arr.append(li)
    return arr

def neg_weight_N(a, b):
    N = 2
    w = min(calc_weights(a, b, N))
    while w >= 0:
        N += 1
        w = min(calc_weights(a, b, N))
    return N

def min_weight(a, b, N):
    return min(calc_weights(a, b, N))
```

Листинг 8: Веса

Результат таков:

```python
    print("First N with negative weight:")
    print(neg_weight_N(-1, 1))

First N with negative weight:
9
```

Листинг 9: Результаты

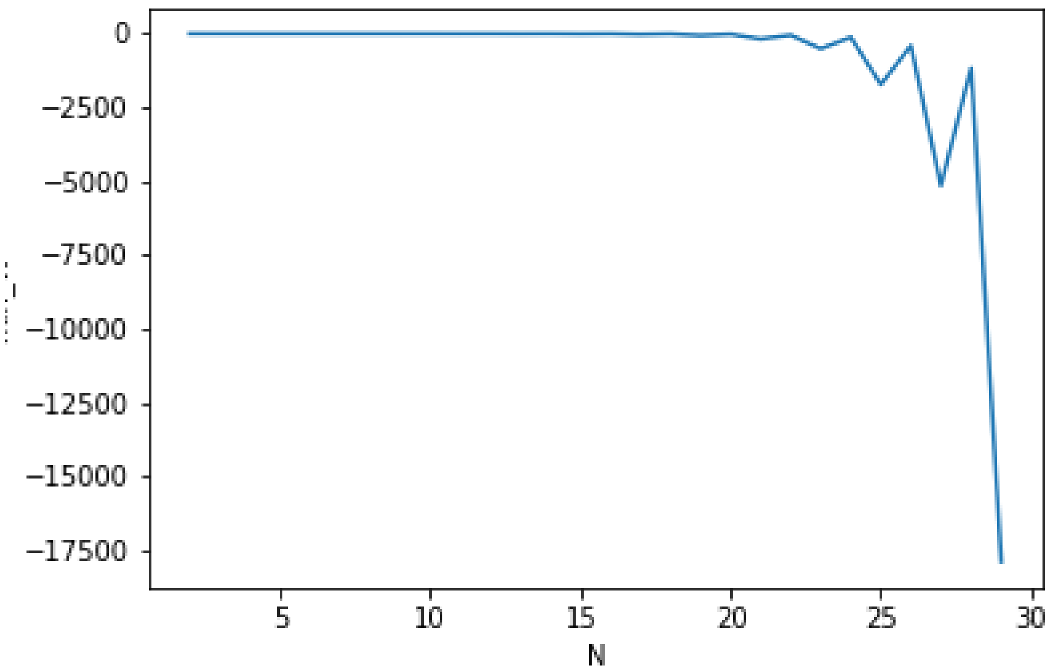
Также легко проверить свойства лямбд, и они оказываются верны.

Полезно также рассмотреть график зависимости минимального веса от числа узлов:

```python
def plot_min_weight(a, b):
    x = []
    y = []

    for n in range(2, 30):
        x.append(n)
```

```
        y.append(min_weight(a, b, n))
    graph.plot(x, y)
    graph.ylabel("min_W")
    graph.xlabel("N")
    graph.savefig("t2c.png")
    graph.close()
```

Листинг 10: Построение графика минимального веса



Получаеся, большие N брать не стоит, так как погрешность сильно вырастет, так как сумма модулей будет большой.