14 Newton–Cotes formulas for integration

14.1 Introductory discussion

Recall that the definite integral of a sufficiently nice function f on the interval $[a, b]$ is denoted

$$\int_a^b f(x)\mathrm{d}x,$$

and has the geometric meaning of a signed area between the graph of f and x-axis. The word "signed" means that I take this area with the sign "plus" if $f(x) > 0$ and with the sign "minus" if $f(x) < 0$. In Calculus we usually use *the fundamental theorem of calculus* to evaluate the definite integral, i.e.,

$$\int_a^b f(x)\mathrm{d}x = F(b) - F(a),$$

where F is any antiderivative of f, i.e., $F'(x) = f(x)$. For instance

$$\int_0^1 e^x \mathrm{d}x = e^x|_0^1 = e^1 - e^0 = e - 1 \approx 1.71,$$

since e^x is an antiderivative of e^x. It is however true that there are functions for which antiderivatives cannot be written down if we are allowed to use only the *elementary* functions: polynomials, rational functions, trig functions, exp and log. Arguably the most famous example is the following integral

$$\int_0^1 e^{-x^2}\mathrm{d}x.$$

It can be proven (not elementary though) that the function $f(x) = e^{-x^2}$ has no *simple* antiderivative. Again, do not get me wrong, all I am saying is that this antiderivative is impossible to write down with our usual supply of functions, not that there is no such thing as the definite integral itself. The matter of the fact is that any antiderivative of e^{-x^2} is not elementary. If you recall your Calculus class, you should be able to write this antiderivative down as

$$F(x) = \int_0^x e^{-s^2}\mathrm{d}s,$$

which, by the second part of the fundamental theorem of calculus, implies that

$$F'(x) = e^{-x^2}.$$

But how to compute F at any given point x? For this I will need to use approximate numerical methods, which you actually already discussed in Calculus class (recall these words: the left point approximation, the right point approximation, the midpoint approximation, etc). These methods were introduced without proper assessment of their accuracy. It is my goal in this section to look at these methods from a somewhat more theoretical point of view.

Math 488/688: Numerical Analysis
e-mail: artem.novozhilov@ndsu.edu. Fall 2021

14.2 Newton–Cotes formulas of order n

Again, the goal is to compute

$$\int_a^b f(x)\mathrm{d}x. \tag{14.1}$$

We know that if we divide the interval $[a, b]$ into n subintervals and compute $n + 1$ points

$$x_j = a + jh, \quad h = \frac{b-a}{n}, \quad j = 0, \ldots, n,$$

then we can compute the Lagrange interpolation polynomial of order n,

$$p_n(x) = \sum_{k=0}^{n} L_k(x) f(x_k),$$

which hopefully will be close to my function:

$$p_n(x) \approx f(x), \quad x \in [a, b].$$

(Remember that this is not always true!)

Taking different n I will get different formulas, specifically,

$$\int_a^b f(x)\mathrm{d}x \approx \int_a^b p_n(x)\mathrm{d}x = \sum_{k=0}^{n} f(x_k) \int_a^b L_k(x)\mathrm{d}x. \tag{14.2}$$

Formulas (14.2) are called the Newton–Cotes formulas of order n. To be able to use them, I need to compute the weights

$$w_k = \int_a^b L_k(x)\mathrm{d}x, \quad k = 0, \ldots, n$$

which *depend* only on the mesh points x_j. Once computed, they can be used for any given function f on the interval $[a, b]$:

$$\int_a^b f(x)\mathrm{d}x \approx w_0 f_0 + w_1 f_1 + \ldots + w_{n-1} f_{n-1} + w_n f_n,$$

where I use the notation $f_j = f(x_j)$, which will be convenient below.

Now let's switch from the general discussion to some concrete examples.

Example 14.1 (Trapezoidal rule)**.** Take $n = 1$ for my interpolation polynomial of order 1 (I cannot take $n = 0$ since I always want to include at least two points $x_0 = a$ and $x_n = b$ but see below). The geometric meaning of this approximation should be clear, vis. I replace the area under the graph of my function f with the area of the trapeze with vertexes at $(x_0, 0), (x_0, f_0), (x_1, f_1), (x_1, 0)$ (make a picture!). Denoting $h = b - a$, I immediately get that the area of this trapeze is $h(f_0 + f_1)/2$, which will be my Newton–Cotes formula of order 1:

$$\int_a^b f(x)\mathrm{d}x \approx \frac{h}{2}(f_0 + f_1).$$

Let me derive the same formula using Lagrange interpolation polynomial.

I have

$$p_1(x) = f_0\frac{x-x_1}{x_0-x_1} + f_1\frac{x-x_0}{x_1-x_0} = f_0\frac{x-x_1}{-h} + f_1\frac{x-x_0}{h}\,.$$

In this case $x_0 = a$ and $x_1 = b$ hence

$$\int_{x_0}^{x_1} p_1(x)\mathrm{d}x = -\frac{f_0}{h}\int_{x_0}^{x_1}(x-x_1)\mathrm{d}x + \frac{f_1}{h}\int_{x_0}^{x_1}(x-x_0)\mathrm{d}x = \frac{h}{2}(f_0+f_1)$$

as expected.

Example 14.2 (Simpson's rule)**.** Now it is time to tackle case $n = 2$. My points now $a = x_0, x_1 = (b+a)/2, x_2 = b$ and $h = (b-a)/2$.

To apply this case I need w_0, w_1, w_2.

Specifically,

$$w_0 = \int_{x_0}^{x_2} L_0(x)\mathrm{d}x = \int_{x_0}^{x_2}\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}\mathrm{d}x = \frac{1}{2h^2}\int_{x_0}^{x_2}(x-x_1)(x-x_2)\mathrm{d}x.$$

I could have computed the last integral directly, but instead I will make the substitution

$$x = ht + x_1.$$

I get (remember that you need to change the expression under the integral sign, the limits of integration, and $\mathrm{d}x$):

$$w_0 = \frac{1}{2h^2}\int_{-1}^{1} ht(ht-h)h\mathrm{d}t = \frac{h}{2}\int_{-1}^{1} t(t-1)\mathrm{d}t = \frac{h}{3}\,.$$

Similarly (an exercise) I can find

$$w_1 = \frac{4h}{3}, \quad w_2 = w_0 = \frac{h}{3}\,.$$

Summarizing, my Newton–Cotes formula of order 2 is given by

$$\int_a^b f(x)\mathrm{d}x \approx \frac{h}{3}(f_0 + 4f_1 + f_2),$$

which is called Simpson's rule (and I hope you recognize this formula since I would guess you saw it in your Calculus class)

Clearly, I can take $n = 3, 4, \ldots$ and the list of Newton–Cotes formulas can be continued. For instance, you are asked in your latest homework to derive the Newton–Cotes formula of order 3:

$$\int_{x_0}^{x_3} f(x)\mathrm{d}x \approx \frac{3h}{8}(f_0 + 3f_1 + 3f_2 + f_3).$$

I have, however, two other related questions. First, of course, is what is the mistake I make when I replace my exact integral with approximate formula. The second question is whether there is a somewhat more efficient method to produce Newton–Cotes formulas for different n and, please note, for possible *not-equally* spaced x_j. Let me start with the error.

Example 14.3 (The error of the trapezoidal rule)**.** Recall that for the Lagrange interpolation polynomial p_n on $[a,b]$ I have the following error estimate

$$|f(x)-p_n(x)| \le \frac{M_{n+1}}{(n+1)!}|\pi_{n+1}(x)|,$$

where, as before, M_{n+1} is the maximum of the $(n+1)$-st derivative of f on $[a,b]$, and

$$\pi_{n+1}(x) = (x-x_0)(x-x_1)\dots(x-x_n)$$

is a polynomial of degree $n+1$. Now, for my specific case $n=1$ I have

$$\begin{aligned}\left|\int_a^b f(x)\mathrm{d}x - \int_a^b p_1(x)\mathrm{d}x\right| &= \left|\int_a^b (f(x)-p_1(x))\mathrm{d}x\right| \le \\ &\le \int_a^b |f(x)-p_1(x)|\mathrm{d}x \le \frac{M_2}{2!}\int_a^b |\pi_2(x)|\mathrm{d}x = \\ &\frac{M_2}{2!}\int_a^b (x-a)(b-x)\mathrm{d}x = (x=ht+a) = \frac{M_2}{2!}\int_0^1 ht(h-ht)h\mathrm{d}t \\ &= \frac{h^3M_2}{12} = \frac{M_2(b-a)^3}{12}\,.\end{aligned}$$

Not surprisingly, in my error estimate for the trapezoidal rule I have the second derivative for the function. Notice that the second derivative of a constant function or polynomial of degree one is zero, which implies that the trapezoidal rule *exact* for any polynomials p_0, p_1. Clearly, this is something I would expect to see. Moreover, it should be quite clear that the Newton–Cotes formula of order n is exact for any polynomial of degree n or below, simply because of the uniqueness of the Lagrange interpolation polynomial, and hence the error estimate for order n must include M_{n+1} and h^{n+2}. For instance, my expectation for the Simpson rule is that the error is proportional to h^4. The reality, however, is better than that! Here are some mathematical details that explain it.

Theorem 14.4. *Consider a Newton–Cotes formula of order n:*

$$\int_a^b f(x)\mathrm{d}x = \sum_{j=0}^n w_j f_j.$$

Then

$$w_j = w_{n-j},$$

and, moreover, if n is even, the resulting approximate formula is exact for any polynomial of degree $n+1$.

Proof. Let $x_j = a+hj, \quad j=0,\dots,n$, $h=(b-a)/n$. I have $x_{n-j} = a+(n-j)h = a+b+hj$, and hence $x_j = a+b-x_{n-j}$.

Now, by definition,

$$w_j = \int_a^b \prod_{i=0,i\ne j} \frac{x-x_i}{x_j-x_i}\mathrm{d}x = (x=a+b-y) = -\int_b^a \prod_{i=0,i\ne j}^n \frac{a+b-y-x_i}{x_j-x_i}\mathrm{d}y.$$

Using the above expression for x_j I find

$$w_j = \int_a^b \prod_{i=0, i\neq j}^{n} \frac{x_{n-j} - y}{x_{n-i} - x_{n-j}} \mathrm{d}y = \int_a^b \prod_{k=0, k\neq n-j}^{n} \frac{y - x_k}{x_{n-j} - x_k} \mathrm{d}y = w_{n-j}$$

as required (note that I relabeled the index in the product).

Now I know that my formula must be exact for any polynomial of degree n. Assume that n is even and consider polynomial

$$q(x) = \left(x - \frac{a+b}{2}\right)^{n+1}.$$

This polynomial has odd degree and antisymmetric with respect to the center of the interval $[a, b]$, meaning that $q(x_j) = -q(x_{n-j})$ and $q(x_{n/2}) = 0$. Therefore,

$$\int_a^b q(x)\mathrm{d}x = \sum_{j=0}^{n} q_j w_j = 0.$$

Finally, any polynomial p_{n+1} of degree $n+1$ can be represented as

$$p_{n+1}(x) = Aq(x) + p_n(x),$$

where A is some constant, therefore the Newton–Cotes formula is exact for any polynomial of degree $n+1$. ■

The discussion above implies that Simpson's rule is exact for any polynomial of degree 3 and below, and hence its error can be estimated as

$$\left| \int_a^b (f(x) - p_3(x))\mathrm{d}x \right| \leq \frac{M_4}{4!} \int_a^b |\pi_4(x)|\mathrm{d}x,$$

the only problem here is that in π_4 I must have 4 points, but Simpson's rules gives me only three: $a, (a+b)/2, b$. Since I am free to choose the last point myself I take it to make the error minimal, specifically, I will take $(a+b)/2$ for the second time, which means

$$\pi_4(x) = (x-a)\left(x - \frac{a+b}{2}\right)^2 (x-b).$$

Since

$$\int_a^b (x-a)\left(x - \frac{a+b}{2}\right)^2 (b-x)\mathrm{d}x = (x = ht - (b+a)/2) = h^5 \int_{-1}^{1} (t+1)t^2(1-t)\mathrm{d}t = \frac{4}{15},$$

I finally get my estimate for the error in Simpson's rule:

$$\left| \int_a^b f(x)\mathrm{d}x - w_0 f_0 - w_1 f_1 - w_2 f_2 \right| \leq \frac{h^5 M_4}{90},$$

i.e., it has the order 5 with respect to the interval length h.

14.3 The method of undetermined coefficients

Is there an easier way to find the approximate formulas? The answer is yes, let me show it by an example.

Example 14.5 (Simpson's rule again)**.** Let $[a, b] = [0, 1]$, and I would like to find the weights to approximate the integral

$$\int_0^1 f(x)\mathrm{d}x = w_0 f_0 + w_1 f_1 + w_2 f_2.$$

In my case I have $x_0 = 0, x_1 = 0.5, x_2 = 1$.

Since I already know that my formula must be exact for any polynomial of degree 2 and below, it have must for exact for specific examples $1, x, x^2$. Certainly I also can compute the integrals from these polynomials, getting:

$$\begin{aligned} 1 &= \int_0^1 1\mathrm{d}x = w_0 + w_1 + w_2, \\ \frac{1}{2} &= \int_0^1 x\mathrm{d}x = \frac{1}{2}w_1 + w_2, \\ \frac{1}{3} &= \int_0^1 x^2\mathrm{d}x = \frac{1}{4}w_1 + w_2, \end{aligned}$$

which is a system of three linear equations with respect to three unknown weights. Solving it I find $w_0 = w_1 = \frac{1}{6}, w_1 = \frac{2}{3}$ and hence

$$\int_0^1 f(x)\mathrm{d}x = \frac{1}{6}\left(f_0 + 4f_1 + f_2\right),$$

which is almost our general Simpson's rule, because all the calculations were done for specific interval $[0, 1]$. Now I would like to change it to an arbitrary interval $[a, b]$. For this task consider a slightly more general problem: how to find a linear change of variables that maps the interval $[c, d]$ (say, of variable t) into the interval $[a, b]$ of variable x? Since

$$x = c(t) = At + B,$$

and the ends of the interval must be mapped into the corresponding ends, I must have

$$b = Ad + B, \quad a = Ac + B,$$

which gives me the solution

$$A = \frac{b-a}{d-c}, \quad B = \frac{ad - bc}{d-c}$$

which allows me to generalize the result I obtained for the interval $[0, 1]$ to an arbitrary $[a, b]$. Namely,

$$\int_a^b f(x)\mathrm{d}x = (x = c(t)) = \frac{b-a}{d-c}\int_c^d f(c(t))\mathrm{d}t \approx \frac{b-a}{d-c}\sum_{i=0}^{n} w_i f(c(t_i)),$$

where w_i were already computed. In my case, $c = 0, d = 1$, and hence

$$\int_a^b f(x)\mathrm{d}x \approx (b-a)\int_0^1 f((b-a)t + a)\mathrm{d}t = (b-a)(w_0 f(a) + w_1 f((a+b)/2) + w_2 f(b)),$$

and using the weights I found above I recover the general Simpson rule.

Similar to what I have done above, I can find now Newton–Cotes formulas for any points, not necessarily equally spaced. For instance there are so-called *open* Newton–Cotes formulas of order n that use the step size $h = (b-a)/(n+2)$ and disregards points $x_0 = a$ and $x_{n+2} = b$ in the analysis. For instance for $n = 0$ one recovers familiar *midpoint* rule

$$\int_a^b f(x)\mathrm{d}x \approx \frac{(b-a)}{2} f((a+b)/2)\,.$$

14.4 Composite Newton–Cotes formulas. Implementation

In the discussion above I used, as examples, only formulas of order 1 (trapezoidal rule) and 2 (Simpson's rule). I only mentioned the formula of order 3, and never discussed the formulas of order 4 and above. The reason for this is that it is quite possible to get a large error by using Lagrange interpolation polynomials of higher order (recall the discussion of interpolating the function $1/(1+x^2)$). What is usually done in practice is not an increase of the order, but division the whole interval $[a, b]$ into smaller intervals and using obtained above formulas for each interval separately.

Example 14.6 (Composite trapezoidal rule)**.** The goal is to compute, as before,

$$\int_a^b f(x)\mathrm{d}x.$$

Let N be the number of intervals I divide $[a, b]$ into, hence I have $N + 1$ points total, and the length of each interval is

$$h = \frac{b-a}{N}\,.$$

Now I apply the trapezoidal rule first on the interval $[x_0, x_1]$:

$$T_1 = \frac{h}{2}(f_0 + f_1),$$

then on the interval $[x_1, x_2]$:

$$T_2 = \frac{h}{2}(f_1 + f_2),$$

and so on, until I get to the final interval $T_N = \frac{h}{2}(f_{N-1} + f_N)$.

Now I consider the sum

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{j=1}^{N} T_j = h\left(\frac{f_0 + f_N}{2} + f_1 + \ldots, f_{N-1}\right) = T(N),$$

since each data point f_j is present twice for all $j \neq 0, N$. Since at each interval I make an error that does not exceed $M_2 h^3/12 = M_2(b-a)^3/(12N^3)$ and there are N intervals, my total error is

$$\left|\int_a^b f(x)\mathrm{d}x - T(N)\right| = \mathcal{O}\left(\frac{(b-a)^3 M_2}{N^2}\right) = \mathcal{O}\left(\frac{1}{N^2}\right),$$

i.e., it is a formula of order $1/N^2$.

Example 14.7 (Composite Simpson's rule)**.** Similarly to the above, I can divide my interval $[a, b]$ into $N = 2k$ intervals (strictly speaking the requirement that the number of intervals if even is not necessary) and apply Simpson's rule to pairs of adjacent intervals. I.e.,

$$S_1 = \frac{h}{3}(f_0 + 4f_1 + f_2), \quad S_2 = \frac{h}{3}(f_2 + 4f_3 + f_4), \quad \ldots, \quad S_{N/2} = \frac{h}{3}(f_{N-2} + 4f_{N-1} + f_N),$$

which, after some simplification, gives

$$\int_a^b f(x)\mathrm{d}x \approx S(N) = \frac{h}{3}(f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \ldots + f_N).$$

Since for each pair of intervals the error has order 5, the total error for Simpson's method can be estimated as

$$\left| \int_a^b f(x)\mathrm{d}x - S(N) \right| = \mathcal{O}\left(\frac{(b-a)^5 M_5}{N^4} \right) = \mathcal{O}\left(\frac{1}{N^4} \right),$$

i.e., Simpson's composite formula has order $1/N^4$ or h^4, as you are asked to confirm numerically in your homework.

The analysis above indicates that for sufficiently smooth function f composite formulas can be made as precise as required given that sufficiently small step h is chosen (but please remember about the roundoff error possibility, which will actually grow with increasing N). Now the final question I would like to answer here is a possible implementation. For my test function I will take very simple

$$f(x) = e^x$$

on the interval $[0, 1]$. Direct calculation immediately gives me

$$\int_0^1 e^x \mathrm{d}x = e - 1 \approx 1.718281828459045$$

First, I will present a direct naive approach to implement the composite trapezoidal rule.

```
import numpy as np

#my test function
def f(x: float):
    return np.exp(x)

#composite trapezoidal rule, version 1

def trapez_v1(f, a, b, n):
    '''computes an approximation of the integral of f from a to b using
    the composite trapezoidal rule with n intervals'''

    s = 0
    h = (b - a)/n

    for i in range(1, int(n)):
```

```
        x = a + i * h
        s += f(x)

    return (s + (f(a) + f(b))/2)*h

#a few results
>>>trapez_v1(f, 0 , 1, 10)
1.7197134913893146
>>>trapez_v1(f, 0 , 1, 100)
1.7182961474504175
>>>trapez_v1(f, 0 , 1, 1e3)
1.7182819716491962
>>>trapez_v1(f, 0 , 1, 1e5)
1.7182818284733654
```

It clearly works, however, the major drawback of my function is that it requires the number of intervals as the input parameter. How do I know which n to submit to my function? I do have some theoretical estimates, but they are usually difficult to apply in many concrete cases, and, moreover, they give a very conservative estimate of the error I am making, and my goal is to possibly minimize the number of evaluations of my function. The basic idea, of course, is to estimate my integral for two different n and if the results are sufficiently close to each other, stop computations. Here is how it can be done rather efficient.

Let me consider a sequence of approximations for the same interval $[a, b]$, which I will index by variable K, such that at K-th step I have $N = 2^{K-1}$ intervals. For $K = 1$ I have one interval, and my approximation will be

$$T(1) = h_1 \frac{(f(a) + f(b))}{2}, \quad h_1 = b - a.$$

For $K = 2$ I have 2 intervals, $h_2 = (b - a)/2$, and

$$T(2) = h_2 \left(\frac{f(a) + f(b)}{2} + f((a + b)/2) \right) = \frac{T(1)}{2} + h_2 f((a + b)/2).$$

One more step, $K = 3$, $N = 4$, $h_3 = (b - a)/4$,

$$T(3) = h_3 \left(\frac{f_0 + f_4}{2} + f_2 + f_1 + f_3 \right) = \frac{T(2)}{2} + h_3(f_1 + f_3).$$

Now the idea should be clear: at each next step to compute $T(K)$ I can use the precomputed value $T(K - 1)$ and values of the function f at "newly added" points, whose number is exactly 2^{K-2}, and the most left has the coordinate $a + h_K$ and most right has coordinate $b - h_K$. This will allow me to save a lot of time reusing the results of previous computations. Here is my function that returns $T(K)$ as an improvement over the previous computation $T(K - 1)$, which must be known beforehand.

```
def trp(f, a, b, k, s = 0.0):
    '''returns the approximation of integral of f from a to b
    by the trapesoidal rule as an improvement for the previous value s,
    which was calculated for 2 ** (k - 2) intervals
    '''
```

```
    if k == 1:
        return (b-a)*(f(a) + f(b))/2

    n = 2 ** (k - 1) #number of intervals
    h = (b - a)/n  #stepsize
    x = np.linspace(a + h, b - h, 2 ** (k - 2)) #points at which calculate f

    return s / 2 + h * np.sum(f(x))
```

Here I use the functionality of `numpy` package, which is very well suited to work with vectorized arguments. In particular, I use the function `np.linspace(A,B,N)`, which returns array of `N` values from `A` to `B`, equally spaced between `A` and `B`. I also simply evaluate the function `f` on the array `x`, which produces a very concise and clean code (cf., with realization above). Now that I have my auxiliary function `trapezoidal`, I can actually implement a somewhat more efficient routine.

```
def trapez_v2(f, a, b, tol = 1e-8, max_iter = 25):
    '''computes an approximation of integral of f from a to b using the
    composite trapezodal rule with given tolerance. The computation stops
    if relative error is subsequent computations is less than
    given tolerance. The maximum number of intervals is 2 ** (max_iter-2)'''

    old = -1e30 #something very unlikely to be close to an answer

    for i in range(1, max_iter):
        new = trp(f, a, b, i, old)
        if np.abs(old - new) < tol * np.abs(old):
            return new
        old = new

    raise Exception('Max number of iterations is reached')

#result
>>> trapez_v2(f,0,1)
1.7182818305927445
>>> trapez_v2(f, 0, 1, 1e-11)
1.718281828461129
```

Can we do better? And actually there are quite deep theoretical reasons to see how a more efficient code can be written. The truth is that when one approximates an integral with a composite trapezoidal rule, the actual error has the following representation:

$$\int_a^b f(x)\mathrm{d}x = T(N) + c_1 h^2 + c_2 h^4 + c_3 h^6 + \ldots,$$

note the absence of odd powers of h. Similarly, one has

$$\int_a^b f(x)\mathrm{d}x = T(N/2) + c_1 (2h)^2 + c_2 (2h)^4 + c_3 (2h)^6 + \ldots,$$

because decreasing the number of intervals twice increases the step twice. Denoting the exact value

of integral I I get

$$I = \frac{4T(N) - T(N/2)}{3} + \tilde{c}_1 h^4 + \ldots,$$

that is the formula

$$I \approx \frac{4T(N) - T(N/2)}{3}$$

has the fourth order, and it is computed by the trapezoidal rule with number of intervals, which is doubled at each step, as it was done in my code above. Hence, I can write new function that uses this advantage. Specifically,

```
def trapez_v3(f, a, b, tol = 1e-8, max_iter = 25):

    old = old_s = -1.0e30

    for i in range(1, max_iter):

        new = trp(f, a, b, i, old)
        new_s = (4.0 * new - old)/3.0
        if np.abs(new_s - old_s) < tol * np.abs(old_s):
            return new_s

        old = new
        old_s = new_s

    raise Exception('Max number of iterations is reached')

#some results
>>>trapez_v3(f, 0, 1, 1e-11)
1.7182818284591843
```

In particular, the version `trapez_v3` usually requires two times fewer computations compared with `trapez_v2` and is a sufficiently robust numerical integrator for not that complicated functions.