Some remarks on quadrature formulæ

Stefano Piani

November 23, 2020

Numerical Integration

Given $f(x) \in \mathcal{C}^0([a, b])$, find a way to compute

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

Idea: Replace $f(x)$ with $\tilde{f}(x)$ so that

- $\tilde{f}$ approximates f (in some sense)
- we know the exact value of $\int_a^b \tilde{f}(x)\,\mathrm{d}x$

Then we may approximate

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

You have studied two ways for approximating a function:

- polynomial interpolation
- L^2–projection on a simpler (polynomial) space

Unfortunately, L^2–projection requires to be able to compute some integrals, so it is not useful.

Therefore, we are forced to use interpolation.

Warning: Interpolation is a sneaky tool!

Let’s change the point of view.
Integrals are somehow a “sum of infinite points”. Therefore, maybe we can approximate them using a sum of “a lot of points”.
We would like to find a family of nodes $x_0, \dots, x_n \in [a, b]$ and a family of weights $w_0, \dots, w_n \in \mathbb{R}$ such that

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{i=0}^{n} w_i f(x_i)$$

A particular choice of x_i and w_i is called *quadrature formula*.

Let us suppose that $a = 0$ and $b = 1$.
We choose $x_0 = 0$ and $x_1 = 1$. We also choose $w_0 = w_1 = \frac{1}{2}$.

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{1}{2} f(0) + \frac{1}{2} f(1)$$

This quadrature formula is called “trapezoidal rule”.

Again, we suppose that $a = 0$ and $b = 1$.
We choose $x_0 = 0$, $x_1 = \frac{1}{2}$. We choose (as before) $w_0 = w_1 = \frac{1}{2}$.

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{1}{2} f(0) + \frac{1}{2} f(\frac{1}{2})$$

How do we choose the nodes? How do we choose the weights?

Exercise: Prove that, if you want a quadrature formula that can integrate correctly the constants, you must have

$$\sum_{i=0}^{n} w_i = b - a$$

We go back to the interpolation. If we choose some nodal points $x_0, \ldots, x_n$ we have that

$$\tilde{f}(x) = \sum_{i=0}^{n} f(x_i) l_i(x)$$

where $l_i(x)$ is the i–th Lagrange polynomial (i.e. the lowest degree polynomial such that $l_i(x_i) = 1$ and $l_i(x_j) = 0$ for $i \neq j$).

Therefore

$$\int_a^b \tilde{f}(x)\,\mathrm{d}x = \int_a^b \left(\sum_{i=0}^{n} f(x_i) l_i(x) \right) \mathrm{d}x = \sum_{i=0}^{n} f(x_i) \int_a^b l_i(x)\,\mathrm{d}x$$

If we define

$$w_i \overset{\text{def}}{=} \int_a^b l_i(x)\,\mathrm{d}x$$

the previous equation becomes

$$\int_a^b \tilde{f}(x)\,\mathrm{d}x = \sum_{i=0}^{n} w_i f(x_i)$$

which is a quadrature rule!

Therefore, a possible way to choose w_i is using the interpolation. But this is not the only possibility!

Moreover, we still miss an idea for choosing the nodes x_i.

A quadrature formula such that its weights come from interpolation is called *interpolatory quadrature formula*.

The trapezoidal quadrature formula is an interpolatory formula. Indeed, if $a = 0$, $b = 1$, $x_0 = 0$ and $x_1 = 1$, we have that

$$l_0 = 1 - x \qquad\qquad l_1 = x$$

and therefore

$$w_0 \overset{\text{def}}{=} \int_a^b l_0(x)\,\mathrm{d}x = \frac{1}{2} \qquad w_1 \overset{\text{def}}{=} \int_a^b l_1(x)\,\mathrm{d}x = \frac{1}{2}$$

which are exactly the weights of the trapezoidal rule.

The quadrature formula of the second example (on the interval $[0, 1]$, we choose $x_0 = 0$, $x_1 = 1/2$ and $w_0 = w_1 = 1/2$) is *not* an interpolatory formula.
Indeed, for an interpolatory quadrature formula we have that

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

where $I_n(f)$ is the result of apply the the quadrature formula on $f(x)$.

$$I_n(f) \overset{\text{def}}{=} \sum_{i=0}^{n} w_i f(x_i)$$

If we choose f of degree less or equal than 1 we have that

$$f = \tilde{f}$$

because we are interpolating using two points.
In other words, on a polynomial p of degree 1 we should have that

$$\int_a^b p(x)\,\mathrm{d}x = w_0 p(x_0) + w_1 p(x_1)$$

without any error.

But for $p \overset{\text{def}}{=} x + 1$ we have that

$$\int_a^b p(x)\,\mathrm{d}x = \frac{3}{2}$$

$$w_0 p(x_0) + w_1 p(x_1) = \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \frac{3}{2} = \frac{5}{4}$$

So this quadrature formula is not an interpolatory one.

It is worth noting that, if p would have been a constant k, then we would had

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

$$w_0 p(x_0) + w_1 p(x_1) = \frac{1}{2} \cdot k + \frac{1}{2} \cdot k = k$$

without any error.

Therefore, that quadrature rule is able to integrate exacly all the constants, but not the polynomials of degree one.

Given a quadrature rule, the greatest integer d such that, for every polynomial p of degree less or equal than d, we have that

$$\sum_{i=0}^{n} w_i p(x_i) = \int_a^b p(x)\,\mathrm{d}x$$

is called *degree of accuracy* of the quadrature formula.

For example, the degree of accuracy of the previous quadrature formula is 0. The degree of accuracy of the trapezoidal quadrature rule is 1.

Theorem: Every interpolatory quadrature rule I_n with nodes $x_0, \ldots, x_n$ has degree of accuracy of *at least* n (be aware that the number of nodes is $n+1$).

Are there other methods to compute the weights?

A *composite* quadrature formula is a quadrature rule that has been obtained “repeating” a simpler quadrature rule on some sub–intervals that are a partition of the original one.

Let us consider the following interpolatory quadrature formula. On an interval $[a, b]$, it uses only one node which is $x_0 = a$. We have only one lagrangian polynomial, and it has degree 0. Indeed

$$l_0 = 1$$

and, therefore

$$w_0 = \int_a^b l_0(x)\,\mathrm{d}x = b - a$$

We want to use this quadrature rule to generate a composite formula on the interval $[0, 1]$. We split the interval in two pieces of the same length.

For the interval $[0, 1/2]$ we have that

$$x_0^{(0)} = 0 \qquad w_0^{(0)} = 1/2 - 0 = 1/2$$

For the interval $[1/2, 1]$ we have that

$$x_0^{(1)} = 1/2 \qquad w_0^{(1)} = 1 - 1/2 = 1/2$$

Our composite rule will have two nodes and two weights:

$$x_0 = 0 \qquad w_0 = 1/2$$

$$x_1 = 1/2 \qquad w_1 = 1/2$$

We have found the quadrature rule of the previous examples!

Time to code!

```
for (i = 0; i < 3; i++) {
    unsigned int cp_count;
    unsigned int len = n;
    if (user(group_info[i]){
        return -EFAULT;
    grouplist += NGS_PER_BLOCK;
    count -= cp_count;
```

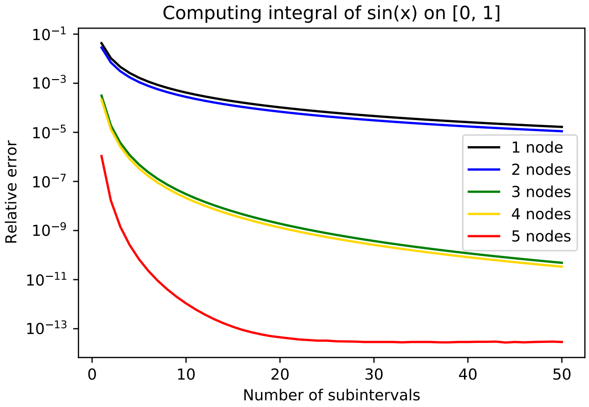
```
group_info = kmalloc(usr)
if (!group_info)
    return NULL;
group_info->ngroups = gidsetsize;
group_info->nblocks = nblocks;
atomic_set(&group_info->usage, 1)
```

```
if (gidsetsize <= NGROUPS_SMALL)
    group_info->blocks[0] = gr;
else {
    for (i = 0; i < nb; i++) {
        gid_t *b;
        b = (void *)__get(GFP);
        if (!b)
            goto partial_alloc;
        group_info->blo[i] = b;
    }
```

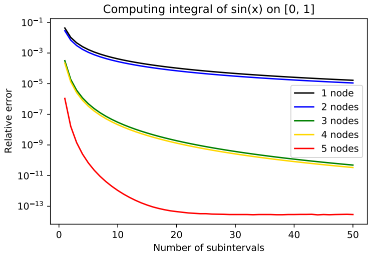
```
struct group
int nbloc
```

```
void groups_free(struct*) {
    if (group_info->blocks[0]) {
        int i;
        for (i = 0; i < 1; i++)
            free_page();
    free(group_info);
```

Using Python, we have obtained this plot:

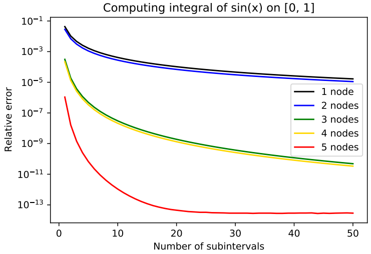


It shows what happens to the error as soon as we split the original interval $[0, 1]$ in some subintervals and we apply a specific Newton–Cotes quadrature rule (interpolatory rules with equally spaced nodes) to each subinterval.



Of course, it shows that using more points for the interpolatory quadrature rule increases the precision.

But there is more: the point here is that increasing the points increases also the speed in which the error decreases. Indeed, the red line decreases much faster than the black one (until it becomes flat because of the numerical errors).



But there is a question that we still need to address: why using 1 and 2 points, for example, we get similar results while adding a third point makes such a difference?

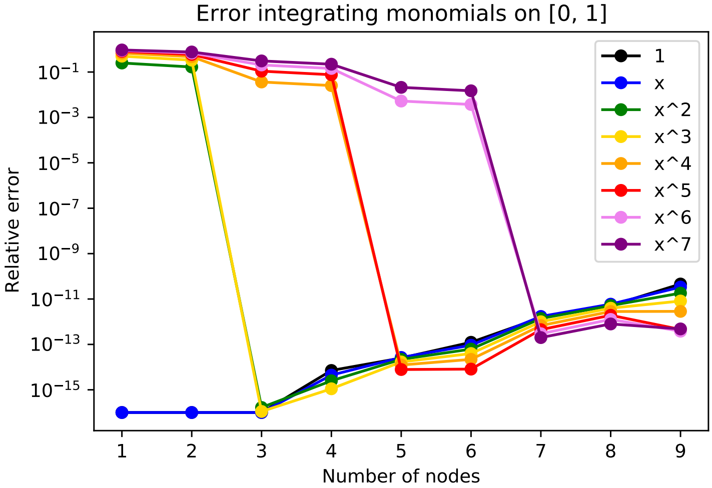
The same is true for 3 and 4 points (that behave in a similar way) compared to 5 that decreases a lot faster.

To find an answer to this question, we want to perform an experiment: we take a few monomials

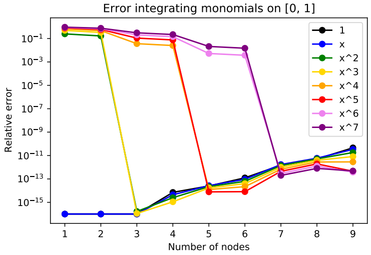
$$1, x, x^2, x^3, \dots, x^7$$

and we try to integrate them on $[0, 1]$ using Newton–Cotes quadrature formulæ with different number of nodes.

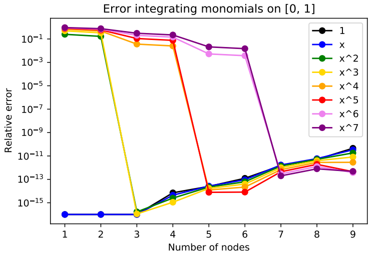
These are the results:



When the error was exactly 0, it has been fixed to 10^{-16} to be visible in the plot.



It is fair to suppose that when the error is of the order of 10^{-12}, the quadrature formula returns the exact result and the error is related to the numerical computation.



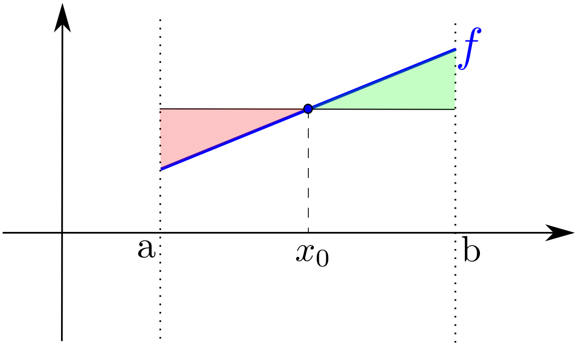
Therefore, there are two question that we must solve:

- Why does each plot increase at the end?
- Why does it drop every two steps?

Why does it increase?
Numerical error: the weights of the Newton–Cotes formulæ are not always positive and, therefore, we create numerical instability. This is important: if we could change the position of the nodes so that the weights would be always positive, the numerical stability of the quadrature formula would increase dramatically.

Why does it drop every two steps?
It seems that using only 1 point, we are able to integrate polynomials of degree 1 (which we would expect that would have required 2 points). At the same time, it seems that we are able of integrating functions of degree 3 with only 3 points. Is this possible?

For what concerns the case with 1 point, what is happening is that we are "mistaking" the polynomial of degree one for a constant that has exactly the same area.



For 3 points, it is more difficult to visualize what is going on.

To check if it is possible that using 3 points we are able to integrate exactly polynomials of degree 3, we take "the most complicated polynomial available" of degree 3 as a test.

Indeed, if our nodes are $x_0, \dots, x_2$, we consider the polynomial

$$w(x) \stackrel{\text{def}}{=} (x - x_0)(x - x_1)(x - x_2)$$

This polynomial is problematic for our quadrature formula. Indeed, if we try to integrate this polynomial with the quadrature formula we have that (now I will call the weights k_i instead of w_i to avoid confusion with the polynomial $w(x)$)

$$I_2(w) = \sum_{i=0}^{2} k_i w(x_i) = 0$$

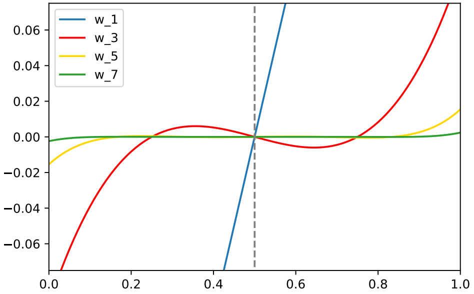
So, the only hope that we have of being able to integrate exactly $w(x)$ is that

$$\int_0^1 w(x)\, \mathrm{d}x = 0$$

Let's have a look at the plots of

$$w(x) \stackrel{\text{def}}{=} (x - x_0)(x - x_1)\cdots(x - x_n)$$

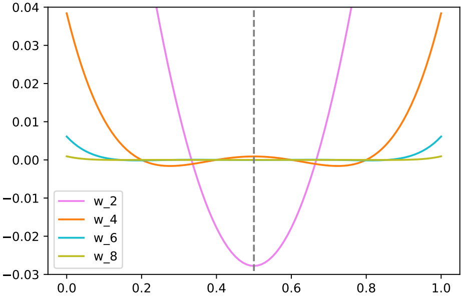
in the interval $[0, 1]$ for different number of equally spaced nodes: 1, 3, 5 and 7; the case we are interested in, 3 nodes, is in red. We added a vertical gray line over $1/2$ to show the symmetry of the polynomials.



If we take a odd number of nodes, the polynomial w is symmetrical and therefore

$$\int_0^1 w(x)\,\mathrm{d}x = 0$$

If we take an even number of nodes, this does not hold anymore



If we use equally spaced nodes, only for an odd number of nodes our quadrature formula is able of integrating without any error the complicated polynomial $w(x)$ (which is called *nodal polynomial*). This is due to the fact that for an odd number of nodes

$$\int_0^1 w(x)\,\mathrm{d}x = 0$$

which is the same result that the quadrature rule will return.

The good news is that if a quadrature formula is able to integrate the nodal polynomial exactly, then it is also able to integrate any other polynomial of the same degree.

Indeed, if we have an interpolatory quadrature formula I_n with nodes $x_0, \ldots, x_n$, $w(x)$ has degree $n+1$. Let us suppose that

$$I_n(w) = \int_a^b w(x)\,\mathrm{d}x$$

If we take $p(x)$ polynomial of degree $n+1$ we have that we can divide p with w obtaining that exists a constant $k \in \mathbb{R}$ such that

$$p(x) = kw(x) + \pi(x)$$

where $\pi(x)$ is a polynomial of degree less or equal than n.

We know that

$$I_n(\pi) = \int_a^b \pi(x)\,\mathrm{d}x$$

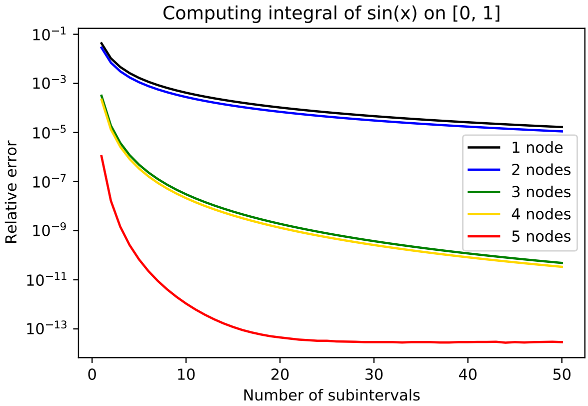
because every interpolatory rule is exact for polynomial of degree less or equal than n.

Then we have

$$\begin{aligned} I_n(p) &= I_n(kw(x) + \pi(x)) \\ &= kI_n(w) + I_n(\pi) \\ &= k\int_a^b w(x)\,\mathrm{d}x + \int_a^b \pi(x)\,\mathrm{d}x \\ &= \int_a^b kw(x) + \pi(x)\,\mathrm{d}x \\ &= \int_a^b p(x)\,\mathrm{d}x \end{aligned}$$

This explains that the Newton–Cotes quadrature formulæ with an odd number of points have degree of accuracy $n+1$, while with an even number of points the degree of accuracy is only n.

Now we can look again at this plot because we have the answer we were missing before:



Why does the formula with 1 node show the same behaviour of the one with two nodes? Because they have the same degree of accuracy (in this case, 2).

This gives us an idea of why the degree of accuracy is important. Indeed, for the Newton–Cotes rules, we have the following theorem.
For every function $f(x) \in \mathcal{C}^\infty([a,b])$ and for every number $n \in \mathbb{N}$, we have that exists a constant K_n (that depends on f and n) such that the composite formula obtained by repeating the Newton–Cotes of $n+1$ nodes on m sub–intervals of the same size has error

$$E \leq (b-a)^{d+1} \frac{K_n}{m^d}$$

where d is the degree of accuracy of the Newton–Cotes formula.

Let us remark the most important consequences of the

$$E \leq (b-a)^{d+1} \frac{K_n}{m^d}$$

- If we increase m we converge to the right result for every Newton–Cotes rule that we are using.
- For a fixed interval, when we increase m the error decreases with a speed that is related only with the degree of accuracy, and not directly with the nodes (so, for example, using Newton–Cotes with 1 or 2 nodes will give us an error that asyntotically decreases with the same speed).

A similar results is also true for the Gaussian quadrature rules.

We have shown that if we use a odd number of nodes for a interpolatory quadrature rule we get a degree of accuracy that is $n + 1$.

Of course, this is true only if the nodes are equally spaced. If we change the position of the nodes, we could loose the simmetry of $w(x)$ and reduce the degree of accuracy to n even for an odd number of nodes.

On the other side, if we move the points in the right way, we may hope to improve the situation.

We know that $w(x)$ is a valid test to check if a quadrature formula with nodes $x_0, \dots, x_n$ reaches the accuracy of $n+1$.

What polynomial can we use to check if we can reach accuracy of $n+2$? The answer is (for example) $x \cdot w(x)$: if

$$\int_a^b w(x)\,\mathrm{d}x = 0 \qquad\qquad \int_a^b x w(x)\,\mathrm{d}x = 0$$

then our formula has accuracy of at least $n+2$.

To reach $n+3$, we need to check also if

$$\int_a^b x^2 \cdot w(x)\mathrm{d}x = 0$$

and so on.

This introduces the following

Theorem: A quadrature formula with nodes $x_0, \dots, x_n$ has degree of accuracy $n+m$ for $m > 0$ if and only if for every polynomial p of degree less or equal than $m-1$ we have

$$\int_a^b p(x) \cdot w(x) \, \mathrm{d}x = 0$$

Of course, there is a limit. We can not hope to take two nodes and to have a formula which is able to integrate exactly all polynomials up to degree 1000. Indeed, we have that

Theorem: A quadrature formula with nodes $x_0, \ldots, x_n$ has degree of accuracy less or equal than $2n + 1$.

Indeed, for every quadrature formula I_n we can consider the polynomial $w(x)^2$ of degree $2n + 2$

$$w(x)^2 = (x - x_0)^2 (x - x_1)^2 \cdots (x - x_n)^2$$

Of course, $I_n(w^2) = 0$ because $w(x)^2$ is 0 on every node. But, on the other side, the integral of $w(x)^2$ can not be 0 because $w(x)^2$ is always non negative (and it is different from 0 in all points beside $x_0, \ldots, x_n$). Therefore, no quadrature rule can integrate exactly $w(x)^2$.

We have therefore proved that reaching a degree of accuracy of $2n+2$ is impossible. But what about $2n+1$?

In this case, it always exists one (and only one) choice of the nodes $x_0, \ldots, x_n$ such that the interpolatory quadrature formula reaches a degree of accuracy equal to $2n+1$ (which is the maximum possible).

An interpolatory quadrature formula that has degree of accuracy equal to $2n+1$ is called *Gaussian quadrature formula*.