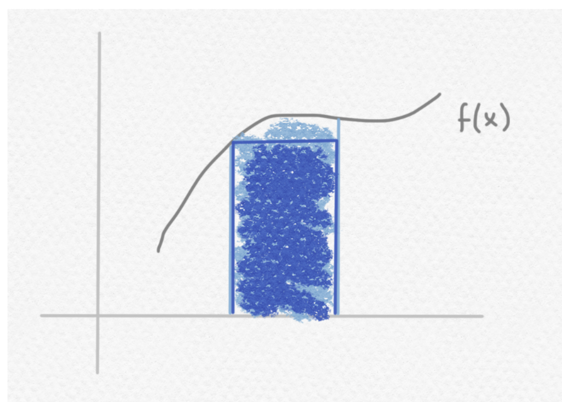
5 Quadrature methods on subintervals

A lot of quadrature methods (i.e. methods for numerically approximating integrals) follow the following pattern. First, we find a way of approximating integrals of a function on *very short intervals* as some weighted combination of function values on that interval. Then, if we want to approximate integrals of a function on a longer interval, we break that large interval up into many small intervals, and then approximate the integrals on each of those subintervals using our method that works well on small intervals.

Here's the simplest possible example of this. To approximate the integral of a function $f(x)$ on an interval $[a, b]$, we could just use the area of the rectangle whose upper-left corner touches the graph of $y = f(x)$ at the point with $x = a$.



In other words:

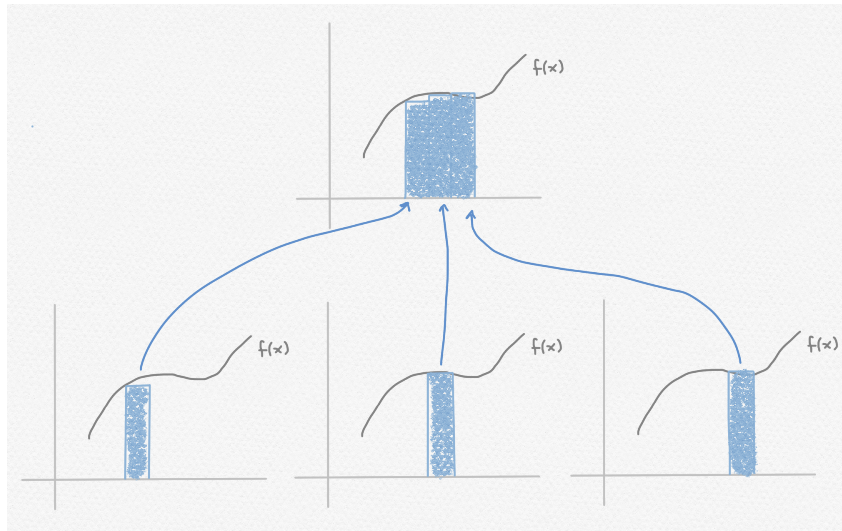
$$\int_a^b f(x)\ \mathrm{dx} = (b - a) \cdot f(a)$$

Of course, this approximation is not necessarily very good. The error can be bounded as follows, using Taylor's Theorem (let me know if you'd like to see exactly how):

$$\left| \int_a^b f(x)\ \mathrm{dx} - (b - a) f(a) \right| \leq \frac{(b - a)^2}{2} |f'(a)|$$

Notice that as the size of the interval shrinks - that is, as $(b - a)$ approaches zero - the size of the error bound shrinks a lot faster than the size of the interval. In particular, the value of the integral is $O(b - a)$, whereas the error bound shrinks like $O((b - a)^2)$. So for very small intervals, we would usually expect the relative error to shrink, but this isn't always useful to us, because we often want to approximate integrals on intervals that aren't tiny. So how can we do this?

The way we accomplish this is by dividing a larger interval into many smaller intervals, and then approximating the integral on each of the smaller intervals, which we're able to do much more accurately. Once we have each of these approximations, we can sum them up to get a decent approximation of the interval on the larger interval. Like this:



We can also show analytically that increasing the number of subintervals (and therefore decreasing their size) will give us better and better approximations. If we use n equally-spaced subintervals, then the length of each interval will be equal to $h = (b - a)/n$. The kth interval will be from $x = a + (k - 1)h$ to $x = a + kh$. Using the error bound mentioned earlier:

$$\left| \int_{a+(k-1)h}^{a+kh} f(x)\, \mathrm{dx} - hf(a + (k-1)h) \right| \leq \frac{h^2}{2} |f'(a + (k-1)h)|$$

When bounding the error of this integral approximation, it's a little cumbersome to keep track of all the derivative values of f at each of the interval endpoints. Instead, we often suppose that we have some bound on the derivative of f that applies *everywhere* in the larger interval $[a, b]$. For instance, if we know that f' is less than M in magnitude everywhere in $[a, b]$, then we have that

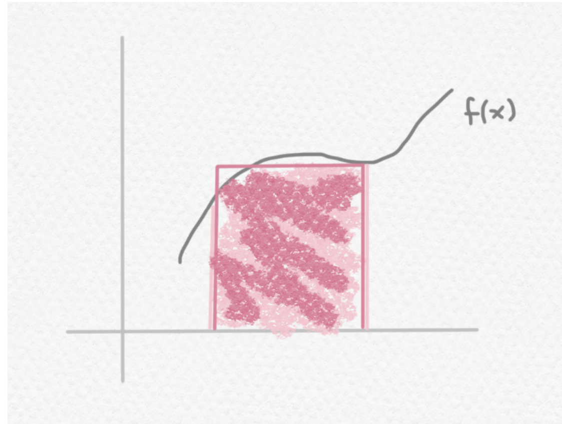
$$\left| \int_{a+(k-1)h}^{a+kh} f(x)\ \mathrm{dx} - hf(a+(k-1)h) \right| \leq \frac{h^2 M}{2}$$

The above tells us that we can expect our approximation of each of the n pieces to have an error of at most $h^2 M/2$. But since there are n pieces overall, we can expect the *overall* error in our approximation (which we obtain by summing up all of the smaller approximations) to be at most $n \cdot (h^2 M/2)$, which is the same as $(b-a)^2 M/2n$ (using the fact that $h = (b-a)/n$). This kind of approximation is also known as an **left Riemann sum**. If we use $\text{LRS}(a, b, f, n)$ to denote the left Riemann sum approximation of the function f on the interval $[a, b]$ with n subintervals, then we've just seen that

$$\left| \int_{a}^{b} f(x)\ \mathrm{dx} - \mathsf{LRS}(a, b, f, n) \right| \leq \frac{(b-a)^2 M}{2n}$$

where M is the maximum magnitude of the derivative of f on the interval $[a, b]$.

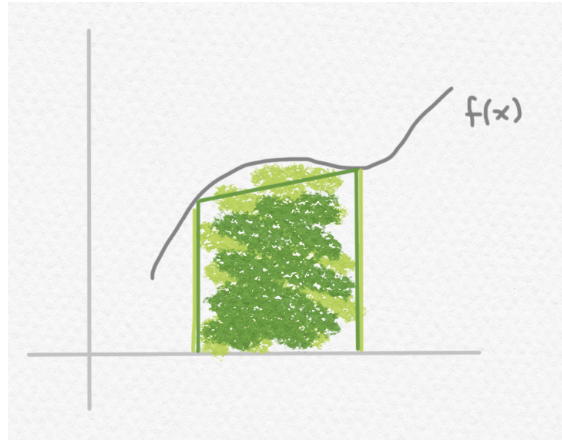
But notice that everything we've done so far has depended on our (kind of arbitrary) decision to approximate the integral of f on small intervals using the area of a rectangle whose height was equal to the value of f at the left endpoint. We could just as easily have chosen to approximate integrals on small intervals using a rectangle whose height is the value of f at the *right* endpoint, like this:



If we follow the same reasoning as before - splitting up a larger interval of integration into much smaller subintervals, and applying this (mediocre) method of approximation on each of those subintervals, we will actually obtain the exact same error bound. When we apply this method on subintervals, what we're computing is called a **right Riemann sum**. If $\text{RRS}(a, b, f, n)$ denotes the right Riemann sum approximation of the function f on the interval $[a, b]$ with n subintervals, we can prove that

$$\left| \int_a^b f(x) \, \mathrm{dx} - \mathsf{RRS}(a, b, f, n) \right| \leq \frac{(b-a)^2 M}{2n}$$

using almost the exact same reasoning. However, by getting more creative with the approximation method we use on *small intervals*, we can actually come up with even more effective **composite methods**. (A "composite method" is what you call a quadrature method that consists of breaking up a large interval into small subintervals, and then applying a certain fixed method on each of those small intervals.) For instance, rather than just using the area of a rectangle, we could use the area of a trapezoid like this:



Asymptotically, this actually does a lot better than the rectangle approximations on small intervals. In particular, we can say that

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

in such a way that the error is guaranteed to shrink as follows:

$$\left| \int_a^b f(x) \, \mathrm{dx} - (b-a) \frac{f(a) + f(b)}{2} \right| \leq \frac{(b-a)^3}{6} |f''(a)|$$

which allows us to conclude (skipping some steps) that the composite rule $\mathrm{TR}(a, b, f, n)$, also called the **trapezoid rule** with n subintervals, gives us an error bounded by

$$\left| \int_a^b f(x) \, \mathrm{dx} - \mathrm{TR}(a, b, f, n) \right| \leq \frac{(b-a)^3 M}{6n^2}$$

where M is the maximum magnitude of the *second* derivative of f on the interval $[a, b]$.

In class and on the homework, you may have seen lots of different quadrature methods such as Simpson's rule, Boole's rule, the Gauss-Lobatto rule and a whole family of different rules called the Newton-Cotes quadrature rules. Each of these rules basically arises by considering a different way of approximating integrals of a function on very small intervals, and then converting it into a *composite rule* by breaking up large intervals into smaller ones and applying the rule to each of the small subintervals. For instance:

- Whereas the trapezoid rule essentially uses linear interpolation on two of a function's values on a small interval, **Simpson's rule** uses *quadratic* interpolation, treating the function as approximately equal to a parabola, and approximating its integral as the integral of that parabola.
- **Boole's rule** is similar to Simpson's rule, but it uses interpolation with quartic (degree 4) polynomials on small intervals, rather than quadratic (degree 2) polynomials.
- More generally, the Newton-Cotes rule with m nodes uses polynomial interpolation with a degree m polynomial on a small interval. Generally, the higher the value of m, the more asymptotically accurate this will be on small intervals.

Because each method of approximating integrals on small intervals gives rise to a more useful method (i.e. a *composite rule*) of approximating integrals on larger intervals, we can write a general-purpose MATLAB function that converts a small-interval rule into its corresponding quadrature rule. Here's a short MATLAB routine to accomplish this:

```
function I = subinterval_quad(a, b, f, ns, mtd)
    I = [];
    for n=ns
        h = (b-a)/n;
        I(end+1) = sum(mtd(a+h*(0:(n-1)), h, f));
    end
end
```

Here's what the arguments of this function represent:

- `a` is the left endpoint of the interval of integration, and `b` is the right endpoint
- `f` is the function we want to integrate
- `ns` is a list of values specifying how many subintervals we want to split the large interval into - for instance, if we just want to get one approximation with `10` subintervals, we could use `ns = [10]`, but if we wanted to do a series of better and better approximations with more and more subintervals, we could pass something like `ns = 2.^(1:20)`
- `mtd` should itself be a function, representing the small-interval technique we want to use - keep reading for examples

Notice how for each `n` value provided, this function calculates the lengths `h` of the subintervals, and then sums up the approximations on those subintervals as given by `mtd`, which would be

provided as an argument. For instance, if we wanted to implement left Riemann sums, we could use the following simple function as our `mtd` argument:

```
function v = lrs_meth(a, h, f)
    v = h*f(a);
end
```

whereas for right Riemann sums, we could use

```
function v = rrs_meth(a, h, f)
    v = h*f(a+h);
end
```

and for trapezoid approximations:

```
function v = trapezoid_meth(a, h, f)
    v = h.*(f(a)+f(a+h))/2;
end
```

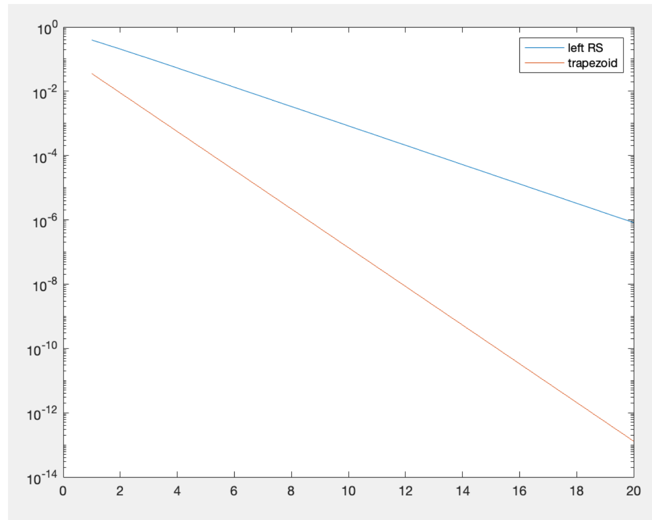
Now, if we wanted to compare how the *left Riemann sums* stack up to the *composite trapezoid rule* approximations, we can use the following code to make a plot:

```
ns = 2.^(1:20);
f = @(x) exp(x)
a = 0
b = 1
I = exp(1) - 1;

lrs_approxs = subinterval_quad(a, b, f, ns, @lrs_meth);
trap_approxs = subinterval_quad(a, b, f, ns, @trapezoid_meth);
lrs_errs = abs(lrs_approxs - I);
trap_errs = abs(trap_approxs - I);

semilogy((1:20), lrs_errs), hold on;
semilogy((1:20), trap_errs);
legend('left RS', 'trapezoid');
```

which gives rise to the following plot:



One of the homework problems asks you to compare a composite Gauss-Lobatto method with the composite Simpson's method. If you want, you could even use the `subinterval_quad` method I've written here to make your work easier - just make sure you understand what it's doing. (But watch out! Implementing Gauss-Lobatto this way might not be as "efficient" as it could be, since it could result in repeated function evaluations. So if you want full credit for an "efficient function", you may have to custom-write a function for GLB4 that does not repeat any evaluations.)