Problem 2 : Adaptive step-size integrator

First, I will explain how the recursive algorithm designed in class works. Then, I will describe the modifications I made to it.

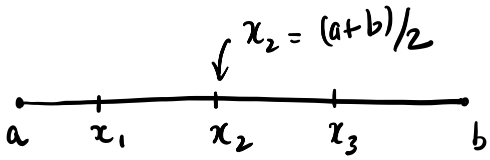
(i) The in-class integrator function (integrate_lazy):

Suppose I'm given an interval $[a,b]$ and a function $f(x)$, $x \in [a,b]$ and I would like to compute

$$I = \int_a^b f(x)dx.$$

Suppose also that I'm given a numerical tolerance "tol".

Step 1: Partition $[a,b]$ into 4 (equal) intervals



Step 2: Evaluate the 3-point (Simpson's Rule) integral using $\{a, x_2, b\}$ (I_1) and the extended integral using $\{a, x_1, x_2 + x_2, x_3, b\}$ (I_2)

If $|I_2 - I_1| < tol$, return I_2 and exit function.

Else if $|I_2 - I_1| >= tol$, go to step 3.

Step 3: Partition $[a,b]$ into two intervals, $[a, x_2]$ and $[x_2, b]$.

Call integrate_lazy() on the two intervals separately, where for $[a, x_2]$, $a = a$ and $b = x_2$ and for $[x_2, b]$, $a = x_2$ and $b = b$, with tol/2 as the tolerance for these new integrals (call them int1 and int2). (For future iterations, both a and b will differ from the original integration limits for some subintervals).

Return the sum int1 + int2 and exit the function.

Step 3 invokes the function recursively, so if after the second iteration, if say int1 does not meet the tolerance requirement, then the interval is further subdivided into two and so on, until the tolerance requirement is met for both int1 and int2 (whose sum will now be the sum of the integral over all the subintervals) and int1 + int2 is finally returned and the recursion stops.

(ii) My modification (integrate_adaptive ()):

- First iteration: I check the length of the "extra" variable. Because of the way I will design my code, if np.size(extra) == 1 that means that extra == None. If this is true, then I divide the interval $[a,b]$ into 4 equal subintervals and call the function at the 5 endpoints of the intervals and do Step 1 from the class algorithm.

If $|I_2 - I_1| < tol$, I again return I_2 as before.

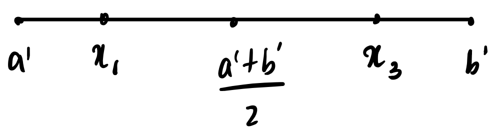
If $|I_2 - I_1| >= tol$, I recursively call integrate_adaptive ( ) with the endpoints modified as before, but now, I change extra to hold the array $[y(a), y(x_1), y(x_2)]$ for the interval $[a, x_2]$ and extra $= [y(x_2), y(x_3), y(b)]$ for the second interval.

I also return the sum of the integrals in case they both meet tolerance requirements.

- <u>Next iterations</u>: Now np.size(extra) = 3 > 1. In this case, I define 2 new x-points: $x_1 = a' + \frac{(b'-a')}{4}$, $x_3 = a' + \frac{3}{4}(b'-a')$

Where a' is the smallest x in said interval and b' is the largest. Note that extra now contains the y-values $y(a')$, $y(b')$ and $y\left(\frac{a'+b'}{2}\right)$

$\boxed{dx' = x_1 - a'}$ ← Modify dx



I then define a new y array $= \left[\overbrace{y(a'), y(b'), y\left(\frac{a'+b'}{2}\right)}^{\text{Taken from extra}}, \overbrace{y(x_1), y(x_3)}^{\text{New evaluations}}\right]$

and repeat Step 1 from class again.

If the tolerance requirement is met, the extended sum is returned.

Else

$[a', b']$ is divided into two intervals again as done earlier and integrate_adaptive() is recursively called on each interval with tol → tol/2 and with extra (interval 1) $= \left[y(a'), y(x_1), y\left(\frac{a'+b'}{2}\right)\right]$ and

extra (interval 2) $= \left[y\left(\frac{a'+b'}{2}\right), y(x_2), y(b')\right]$.

The sum of these two integrals is returned, and the rest of the functioning is the same as the integrator from class.

Some extra mods made later

- I use two global variables to count the number of function calls made by each integrator.
- I check each function call to see if any $y(x_i)$ diverge and print an Error message and replace $y(x_i) \to 0$.