Homework 6
APPM/MATH 4650 Fall '20 Numerical Analysis

Due date: Saturday, October 17, before midnight, via Gradescope. **Instructor**: Prof. Becker
Theme: Numerical integration

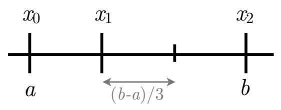
Instructions Collaboration with your fellow students is OK and in fact recommended, although direct copying is not allowed. The internet is allowed for basic tasks (e.g., looking up definitions on wikipedia) but it is not permissible to search for proofs or to *post* requests for help on forums such as `http://math.stackexchange.com/` or to look at solution manuals. Please write down the names of the students that you worked with.

An arbitrary subset of these questions will be graded.

Turn in a PDF (either scanned handwritten work, or typed, or a combination of both) to **Gradescope**, using the link to Gradescope from our Canvas page. Gradescope recommends a few apps for scanning from your phone; see the Gradescope HW submission guide.

We will primarily grade your written work, and computer source code is *not* necessary except for when we *explicitly* ask for it (and you can use any language you want). If not specifically requested as part of a problem, you may include it at the end of your homework if you wish (sometimes the graders might look at it, but not always; it will be a bit easier to give partial credit if you include your code).

Problem 1: Quadrature on non-equispaced nodes Consider (non-composite) quadrature to estimate $\int_a^b f(x)\,dx$ using nodes $x_0 = a$, $x_1 = a + \frac{1}{3}(b-a)$, $x_2 = b$. See the figure below:



Determine a quadrature scheme using these nodes that can integrate quadratic polynomials exactly (i.e., has degree of accuracy 2). *Hint:* Without loss of generality, you can write $h = (b-a)/3$ and then let $x_0 = 0$. *Hint:* Interpolate! *Hint:* Re-use some of the work from last week's HW. *Hint:* Check your work numerically by seeing if your scheme really does integrate a quadratic exactly.

Problem 2: Make a composite version of the scheme you developed in Problem 1, and use it to integrate $f(x) = e^{3x}$ on $[-1, 2]$. Plot the error as a function of the number of nodes n (choose a large value of n, like 10^2 or 10^4; you may want to make sure n is a multiple of a small number); as always, we suggest `logspace` here. Also plot the error using composite midpoint, trapezoidal and Simpson's rule. Using the plot, estimate the order of the new composite rule you created.

Problem 3: Romberg integration Make a Romberg integration scheme applied to the composite *midpoint* rule, and evaluate this to approximate $\int_0^{50} \cos(x)\,dx$ using up to 4096 nodes. Report a table of your error with each row a larger value of n and each column the next extrapolation value for that n (e.g., similar to Tables 4.10 and 4.11 in Burden and Faires, though note these are for composite *trapezoidal* rule). Both composite midpoint and composite trapezoidal rules are $O(h^2)$ (assuming f has bounded derivatives), but is there an advantage to using composite trapezoidal rule in Romberg integration? *Note*: Your code does *not* need to efficiently re-use function evaluations on nodes.

Problem 4: Fourier things Consider a function f which is periodic on $[-\pi, \pi]$. We define the k^{th} **Fourier coefficient** (for $k \in \mathbb{Z}$, where $\mathbb{Z}$ are the integers) as

$$\hat{f}_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) e^{-ikx}\,dx \tag{1}$$

which are then used to represent f as a **Fourier series** $f(x) = \sum_{k\in\mathbb{Z}} \hat{f}_k e^{ikx}$. If you haven't seen Fourier coefficients, they are similar to the **Fourier transform**[1] but for periodic functions; if you haven't seen the Fourier transform, don't worry as you don't need a familiarity with it for this problem, but you ought to see the Fourier transform at some point before you graduate (if you're in a STEM field). Yet another related concept is the **discrete Fourier transform (DFT)** (which is what the FFT computes):

$$F_k = \frac{1}{n}\sum_{j=-n/2+1}^{n/2} f_j e^{-i2\pi jk/n}, \quad k = -n/2+1, -n/2+2, \ldots, 0, \ldots n/2-1, n/2 \tag{2}$$

where $f_j = f(j \cdot h)$ with $h = 2\pi/n$ (note we use $i = \sqrt{-1}$, and j is just an integer index)[2]. The DFT is useful in its own right, but it can also be viewed as an approximation to the Fourier coefficients for $|k| \le n/2$. For this problem, we'll assume n is even to simplify indexing, though similar formulas hold if n is odd as well.

a) Assuming f is 2π periodic so $f(-\pi) = f(\pi)$, and for $|k| \le n/2$, show that the DFT in Eq. (2) is the composite trapezoidal rule (with $n+1$ nodes) applied to the integral for the Fourier coefficients in Eq. (1).

b) Let $f(x) = \frac{1}{1+\cos(x)/2}$, for which the Fourier coefficients can be evaluated in closed form using complex variables and the theory of residues (or via Mathematica, Wolfram Alpha, or sympy). For example,

$$\hat{f}_1 = \frac{1}{2\pi}\int_{-\pi}^{\pi} \frac{e^{-ikx}}{1+\cos(x)/2}\,dx = 2 - \frac{4}{\sqrt{3}}$$

Write code to estimate the value of $\hat{f}_1$ using composite midpoint, trapezoidal and Simpson's rule, and plot the error for each method as a function of n (for using $n+1$ nodes), for reasonable values of n.

c) Recalling that the composite trapezoidal rule is the same as the DFT formula, based on your findings in part (b), do you think there's an advantage to using a higher-order method, like composite Simpson's rule, to approximate the Fourier coefficient? Give some theoretical evidence behind your answer.[3]

Optional problem (Not graded) **Beating Mathematica**. Think of a complicated function F, and then symbolically find the derivative $f = F'$ (using your favorite symbolic math software, like Wolfram Alpha, Mathematica, python with sympy, Matlab with the symbolic package, sage, Maple, etc). Try to make F (and hence f) so complicated that the software *cannot* find the antiderivative of f (or at least not in a reasonable amount of time). Then numerically evaluate the integral of f over an appropriate range $[a, b]$ using your favorite quadrature rule, and check the error (since you know the true value $\int_a^b f(x)\,dx = F(b) - F(a)$).[4]

[1]which is "defined" as $\hat{f}(\omega) = \int_{\mathbb{R}} f(x)e^{-ikx}\,dx$ for any $\omega \in \mathbb{R}$ if $\int_{\mathbb{R}} |f|\,dx$ exists (this is the "definition" you learn in a physics or engineering class), though you can also define the Fourier transform on a larger class of functions by thinking of it as a bounded linear transformation and using density arguments, or by using in a weak sense (using test functions) so that you can even make sense of the Fourier transform of a Dirac delta function; such results are discussed in APPM 5450 "Applied Analysis." Engineers and physicists often implicitly use this extended definition.

[2]For all these transforms, there are many different conventions for the normalization constants, and in the end it's not too important as long as you are consistent, which usually means adjusting the appropriate *inverse* transform.

[3]For further reading on the connections between the DFT and quadrature, see chapter 9 in *The DFT: An Owner's Manual for the Discrete Fourier Transform* by William Briggs and Van Emden Henson, SIAM (1995); the function used in this exercise is based on the similar function on page 366 of that book. For a precise statement on when there is no error at all in evaluating the Fourier coefficient via the DFT, see Theorem 9.3 in that book.

[4]In the case of Mathematica, we haven't really *beaten Mathematica*, just its `Integrate` function, since Mathematica includes quadrature-based numerical integration in `NIntegrate`.