Scientific Computing, Courant Institute, Fall 2021
http://www.math.nyu.edu/faculty/goodman/teaching/SciComp2021/index.html

Second assignment

You should read Chapter 3 of *Principles of Scientific Computing* before doing some of these problems. Some of the material needed for the exercises was not covered in lecture.

1. Find a difference approximation of the form

$$f'(x) = \frac{1}{h}\left[a_0 f(x) + a_1 f(x+h) + a_2 f(x+2h) + a_3 f(x+3h)\right] + O(h^4) \ .$$

   One way to do this is using Richardson extrapolation with three first order estimates $A(h)$, $A(2h)$, and $A(3h)$, with $A(h) = \frac{1}{h}\left[f(x+h) - f(x)\right]$.

2. (*Try to look up the math background for the "big Oh" notation here, but if you haven't worked with it before, your answer may fall short of what a mathematician would consider a proof. Just do your best.*) Suppose R is a region in the plane with a "nice" boundary ("nice" meaning "not too jagged" but not not being defined completely). You can estimate the area of R by counting the number of *lattice points* inside R. Let h be a small parameter and define the h lattice points as $(x_j, y_k) = (hj, hk)$. The lattice point area estimate is

$$A(h) = \frac{1}{h^2} \# \left\{(j,k) \mid (x_j, y_k) \in R\right\} \ .$$

   If $\mathcal{S} = \{\cdots\}$ is a set, then $\# \{\cdots\}$ denotes the number of elements in $\mathcal{S}$. Show that $A(h)$ is first order accurate, $|A(h) - A| = O(h)$, in the sense that there is a C with

$$|A(h) - A| \leq Ch \ .$$

   if h is small enough. Your argument may use the "obvious" geometric fact. Let Γ is the boundary of R, and let $\text{dist}((x,y),\Gamma)$ be the distance from (x,y) to Γ. If $\text{dist}(x,y),\Gamma) > h$, then the little square with side h and center (x,y) is either entirely inside R or entirely outside R. The collection of boxes around the lattice points (x_j, y_k) that are entirely inside R have total area $\leq A$. The collection of boxes that touch R have area $\geq A$. If R is "nice" then

$$B(h) = \# \left\{(j,k) \mid \text{dist}((x_j, y_k), \Gamma) < h\right\} < Ch \ .$$

   Actually, $B(h)$ is something like $\frac{1}{h}\text{length}(\Gamma)$. Next, show that $A(h)$ does not have an asymptotic error expansion. For that, it suffices to take R to be the square

$$R = \text{unit square} = \left\{0 \leq x \leq 1 \,,\ 0 \leq y \leq 1\right\} \ .$$

[(for information only, not part of the exercise) The *circle problem* is to understand $A(h) - A$ when R is the inside of the unit circle. In that case, $A(h) - A = O(h^p)$ for $p > 1$. Numerical experiments suggest it's true for any $p < \frac{3}{2}$. The mathematician Vander Corput proved it's true for $p < \frac{4}{3}$, which is bigger than 1. Don't expect to prove the full $p < \frac{3}{2}$, this implies the Riemann hypotheses (a million dollar math problem, literally).]

3. Exercise 2 from Chapter 3 in the "book". Read about multi-dimensional difference approximations there before trying this.)

4. (*This exercise covers numerics of integration, some aspects of software practice in scientific computing, and more sophisticated adaptive computational algorithms. The sequence of steps illustrates the software development process. You should read the Python notes on classes before starting.*)

   (a) Write a procedure that applies a fixed rectangle rule integration algorithm with n equal sized sub-intervals to estimate

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

   This function that does this should be in a separate module that you can `import` into a main program module. The function should be something like `RR1f(a, b, n, m)`, where `m` is an object that has an attribute `m.f` which is a function of one variable. If `x` is a floating point number, then `m.f(x)` should be a function that return $f(x)$. The name `m` is for "model". The name of the integration function is understood as `RR` for "rectangle rule", `1` for "first order accurate", and `f` for "fixed rule". A sixth order adaptive Gauss quadrature might be called `GQ6a`. Use different naming conventions if you want. Check using a large n and a simple f (e.g., sine or exponential where you can compute the answer in closed form).

   (b) Write a main program that calls your integrator and verifies it converges with first order accuracy as described in the notes. The code can use n as a variable instead of h, so the error expansion for a first order method is $A(n) \sim A + \frac{1}{n}A_1 + \frac{1}{n^2}A_2 + \cdots$. Choose a test problem and a range of n where the first order convergence is clear.

   (c) Modify your integrator from part (a) to do the second order midpoint rule. Use the code from part (b) to verify that the result has asymptotic error behavior appropriate for a second order method.

   (d) Apply the code of part (c) to the integrals

   $$2 = \int_0^1 x^{-\frac{1}{2}}\,dx \ , \quad \text{and} \quad \frac{2}{3} = \int_0^1 x^{\frac{1}{2}}\,dx \ , \quad \text{and} \quad \frac{2}{5} = \int_0^1 x^{\frac{3}{2}}\,dx \ .$$

   What orders of accuracy do you observe? Explain in a mathematical way using the error analysis for the midpoint rule which ones have

convergence rate less than 2. If possible (this is not easy), explain the orders that are less than 2. *Hint.* When you look at error ratios, keep in mind that $2^{\frac{1}{2}} = 1.4142\cdots$, and $2^{\frac{3}{2}} = 2.8284\cdots$.

(e) Write an adaptive code `RR1a(a, b, n0, m, tol)` that starts with $n = n_0$ and continues to double n until asymptotic error analysis suggests that the accuracy criterion $|A(n) - A| \leq$ `tol` is satisfied. The routine should return a "tuple" consisting of the best approximation, the number of points needed to get it, and character string that tells you whether the requested error was achieved or not. The routine should have a bound (possibly named `N_max` so that it will declare failure if it gets to $n > N_{\max}$ without satisfying the error criterion. It should call the fixed n rectangle rule code to do the actual approximate integrations. Find a hard problem where it works, such as

$$\int_0^1 \lambda e^{-\lambda x}\, dx \ .$$

Find a problem where it does not work, such as $f(x)$ being discontinuous and `tol` very small. The output should demonstrate that the calling routine "understood" that the adaptive integration routine did not work.

(f) Apply the code to the problem

$$f(t) = \int_0^1 \cos(tx^2)\, dx \ .$$

Do an experiment to see how many points are needed to compute $f(t)$ accurately for large t. If your code is good and well automated, trial-and-error experiments like this should be easy. See whether you can get into the "asymptotic regime" of t so large that the large t asymptotic expansion applies:[1]

$$f(t) \sim \sqrt{\frac{\pi}{8t}} + \frac{1}{2t}\sin(t) - \frac{1}{4t^2}\cos(t) - \cdots \ .$$

You will appreciate adaptive integration when you look at the numbers n needed for large t.

(g) Repeat part (f) but with a higher order basic integration method. It should be easy and quick to swap out the rectangle rule fixed n function and swap in one using Simpson's rule, or the 3 point sixth order Gauss quadrature rule. This is where you learn whether you wrote good code for the earlier parts. Comment on the computation

[1] This approximation may be found by the "traditional" methods of asymptotic approximation of integrals. The book *Advanced Mathematical Methods for Scientists and Engineers* by Carl Bender and Steve Orszag explains the methods. The first terms is from "near zero" and is the value, properly defined, of the *Fresnel integral* $\int_{-\infty}^{\infty} \cos(tx^2)dx$. The rest of the terms come from "near 1" (the other endpoint) and may be found using integration by parts.

time (n) needed for large t for the low and high order integration methods. For a given $N_{\max}$, you should be able to do much larger t using the high order method without failing, and with a small `tol`. Note that the answer is going to zero, so you might look for relative rather than absolute accuracy in part (e).

(h) There was supposed to be something using the `slice` attribute of the Log Likelihood class, but I think this is enough.