Math 128A, Summer 2019
Lecture 16, Thursday 7/18/2019

Topics Today:

- ECTR
- Gaussian Integration
- Adaptive Integration

1 Midterm Problem 2

This is a model problem for when we want to look at the distances between things like us, satellites and space stations. Strain gives a beautiful explanation that I (in retrospect should have but) did not type up.
The summation

$$\begin{aligned} g_i &= \sum_{j=1}^{n} \frac{f_i}{x_i - y_j}; \qquad O(n^2) \\ &= \sum_{j=1}^{n} \frac{1}{x_i} \sum_{q=0}^{p-1} \underbrace{\left(\frac{y_j}{x_i}\right)^q}_{y_j^{\varepsilon}\left(\frac{1}{x_i}\right)^{\varepsilon}} + O(\varepsilon) \end{aligned}$$

has been proven that we need order of $O(n^2)$ work. For precision to the order of $\varepsilon = 2^{-52}$, it turns out we need $p \geq 52$.
The first rule of applied mathematics is to swap order of sums to see what this gives.
Notice because y is small and x is tiny, we write:

$$\frac{1}{x-y} = \frac{1}{x}\frac{1}{1-\frac{y}{x}} = \frac{1}{x}\sum_{q=0}^{\infty}\left(\frac{y}{x}\right)^p.$$

2 Endpoint Corrected Trapezoidal Rule (ECTR)

The first part of the homework looks at the Euler-Maclaurin summation formula.

$$\sum_{0}^{1} f(t)\ dt = \frac{1}{2}\left[f(0) + f(1)\right] + b_1\left[f'(1) - f'(0)\right] + b_2\left[f'''(1) - f'''(0)\right]$$

When we compound these, we get:

$$\int_0^n f(x)\ dx = \underbrace{\left[\sum_{0}^{n} f(j)\right]''} + b_1\left[f'(n) - f'(0)\right] + b_2\left[f'''(n) - f'''(0)\right] + \cdots$$

$$\sum_{j=0}^{n} f(j) = \frac{1}{2}f(0) + f(1) + \cdots + f(n-1) + \frac{1}{2}f(n)$$

We plug into $f(x)\ dx = e^{-tx}\ dx$, using by-parts: $\frac{d}{dx}e^{-tx}$.

$$\begin{aligned}\int_0^\infty e^{-tx}\ dx = \frac{1}{t} &= \left[\sum_{j=0}^{\infty}\right]'' + b_1\left[-f'(0)\right] + b_2\left[-f'''(0)\right] + \cdots \\ &= \frac{-1}{2} + \frac{1}{1-e^{-t}} + b_1 t + b_2 t^3 + \cdots\end{aligned}$$

Essentially, we are plugging in the function e^{-tx} to 'extract the b's ', because we know if it works for everything, then it must work for our example. Multiplying out by t, we have:

$$t\sum_0^\infty e^{tx}\ dx + \frac{t}{2} = \frac{t}{1-e^{-t}}$$

Then we set the bernoulli numbers b_i:

$$b_1 t^2 + b_2 t^4 + b_3 t^6 + \cdots = 1 + \frac{t}{2} - \frac{t}{1-e^{-t}}$$

The explicit formula for the bernoulli numbers is:

$$\begin{aligned} b_1 &= \frac{B''(0)}{2!} \\ b_2 &= \frac{B''''(0)}{4!} \\ &\vdots\end{aligned}$$

but we don't want to do this kind of math, even if it's finite, because taking the derivatives can be tedious. We do this another way:

$$\int_0^1 f(t)\ dt = \int_0^{nh} f(t)\ dt = h\left[\sum_{j=0}^{n} f(jh)\right]'' + \underbrace{b_1 h^2\left[f'(nh) - f'(0)\right]} + b_2 h^4\left[f'''(nh) - f'''(0)\right] + \cdots$$

The mean value theorem gives us that the underbraced expression is equal to $-\frac{1}{12}h^2 f''(\xi)$, so we know $b_1 = \pm\frac{1}{12}$ (one of these, Strain can't remember). We then toss in everything we know about the error:

$$\int_0^1 f(t)\ dt = h\left[\sum f(jh)\right]'' - \underbrace{\frac{1}{12}h^2\left[f'(nh) - f'(0h)\right]} + O(h^4).$$

We approximate this with the cheapest possible approximation (1 point):

$$f'(nh) = \frac{f(nh) - f([n-1]h)}{h} + O(h^1),$$

whereas if we use two points to make a linear approximation. However, this order of accuracy is not enough, because multiplying into our above expression does not match $O(h^4)$.

Consider

$$f_0' = af_0 + bf_1 + cf_2.$$

A cheap and dirty way to find these coefficients, we try: $f(t) = 1$, $f(t) = t$, and $f(t) = t^2$. If we're lucky, we get three equations and three unknowns.

$$\begin{aligned} 0 &= a + b + c \\ 1 &= a \cdot 0 + b \cdot 1 + c \cdot 2 \\ 0 &= a \cdot 0^2 + b \cdot 1^2 + c \cdot 2^2 \end{aligned}$$

This gives a system of linear equations, netting:

$$a = \frac{-3}{2}, \qquad b = 2, \qquad c = \frac{-1}{2}.$$

This is sufficient to get a fourth-order integration rule. Bringing this back into the main question, we have:

$$\begin{aligned} &h\left[\frac{1}{2}f_0 + f_1 + f_2 + \cdots + \frac{1}{12}\left(\frac{-3}{2}\right)f_0 + \frac{1}{12}(2)f_1 + \frac{1}{12}\left(\frac{-1}{12}\right)f_2\right] \\ &= \frac{h}{24}[9f_0 + 28f_1 + 23f_2 + 24f_3 + 24f_4 + 24f_5 + \cdots \\ &\qquad \cdots + 24f_{n-4} + 24f_{n-3} + 23f_{n-2} + 28f_{n-1} + 9f_n] + O(h^4). \end{aligned}$$

We corrected 3 weights at each end, symmetrically. We only need to do the calculation at one end and read off backwards for the other end.
Compounding is how we get the order of h. By endpoint correcting, we get a much better result than we do in Simpson's rule, which uses quadratic interpolation.

Remark: Strain doesn't actually like ECTR, but it's relatively good when we need to use equispaced points. Using equal weights (except at the endpoints) is the most 'stable' thing we can do, when we do things like fast fourier transforms.

3 Gaussian Integration

As a quick review, recall that our 'recipe' was really simple. Let

$$P_0 := 1, \qquad P_1 := t; \qquad P_{n+1}(t) = t \cdot P_n(t) - c_n$$

Let our points $t_0, \ldots, t_n$ be the roots of the polynomial P_{n+1}, and our weights

$$\begin{aligned} w_j &:= \int_{-1}^{1} L_j(t)\ dt \qquad \text{by hand; } n \leq 4 \\ &= \int_{-1}^{1} [L_j(t)]^2\ dt \end{aligned}$$

Remark: As an aside, when performing bisection, we notice that for polynomials, the roots of some polynomial P_n are 'trapped' between the roots of polynomial P_{n+1}. We say that they are '**interlaced**'.

This result leads us to know that for $t_0^n, \ldots, t_{n-1}^n$ roots of P_n,

$$-1 < t_0^{n+1} < t_0^n < t_1^{n+1} < t_1^n < \cdots < t_{n-1}^n < t_n^{n+1} < 1$$

We take the exact P_n as given by Gaussian Quadrature. The reason this 'interlacing' occurs is due to the fact that P_k is orthogonal to all P_r for all $r < k$. Orthogonal functions are tightly related to integration points, within an interval, for example $[-1, 1]$. If we want to prove the interlacing phenomenon, we can take the differential equations of the Legendre polynomial.
Strain's favorite expression for $w_j = \int_{-1}^{1} L_j(t)\ dt$ is:

$$\begin{aligned} w_j &= \int_{-1}^{1} \lambda_j \frac{\omega(t)}{t - t_j}\ dt \\ &= \lambda_j \int_{-1}^{1} \frac{P_{n+1}(t)}{t - t_j}\ dt \end{aligned}$$

Recall that we defined $\omega(t) =: P_{n+1}(t)$ in class. It turns out that this is an interesting formula. We may not want to use this for numerical analysis (and this class), as we are dividing by $t - t_j$.

4 Adaptive (Gaussian) Integration

The idea is very simple in that we want to capture all the 'wacky' behavior of functions on subintervals. We'll say:

$$\begin{aligned} \int_a^b f(x)\ dx &= \sum_{j=1}^{N} \int_{a_j}^{b_j} f(x)\ dx \\ &= \sum_{j=1}^{n} h_j \end{aligned}$$

Let:

$$\int_a^b f(x)\ dx = h \int_{-1}^{1} f(m + th)\ dt$$

$h_j := \frac{b_j - a_j}{2}$ stands for half-length of the interval. To check if our multiplication by h at the left is correct, we check a function that we know:

$$\int_a^b 1\ dx = b - a = h \int_{-1}^{1} 1\ dt = 2h$$

Hence we write:

$$\begin{aligned} \int_a^b f(x)\ dx &= h \int_{-1}^{1} f(m + th)\ dt \\ &= \underbrace{h_j \sum_{q=0}^{p} w_q f(m_j + t_q h_j)}_{\text{Gaussian Quadrature}} + O\left(h_j^{2p+1} f^{(2p)}(\xi_j)\right) \end{aligned}$$

To see again (differently) how we assigned/defined a, b above, consider the change-of-variables:

$$\begin{aligned} x &= m + th \\ dx &= h \cdot dt \\ t &= -1 \qquad x = m - h =: a \\ t &= 1 \qquad x = m + h =: b. \end{aligned}$$

Then this nets us

$$\int_a^b f(x)\ dx = h \sum_{-1}^{1} f(m + th)\ dt$$

So our main equation is:

$$\int_a^b f(x)\ dx = \sum_{j=1}^{n} h_j \sum_{q=0}^{p} w_q f(m_j + t_q h_j) + O\left(h_j^{2p+1} \underbrace{f^{(2p)}(\xi_j)}_{\text{we don't know}} \right)$$

We choose intervals $[a_j, b_j]$ to subdivide $[a, b]$, where

$$[a, b] = \bigcup_{j=1}^{n} [a_j, b_j]$$

An easier way to do this than manually setting intervals is to define a recursive subdivision of the interval, $I(a, b)$. To do this, we break an interval into:

$$[\ \underbrace{a_1}\qquad \underbrace{b_1 = a_2 = m}\qquad \underbrace{b_2}\ ]$$

Doing so, we get an error estimate of:

$$\text{error} = I(a, b) - [I(a_1, b_1) + I(a_2, b_2)]\,.$$

and we define that if the absolute value of our error estimate is less than a user-specified tolerance, then we're done. If not, then we keep on going (we declare this was progress and keep on subdividing).
One way to do this (computer science) is to build a **stack**; that is, a *list* of intervals:

$$L = (\{[a_1, b_1], I_1\}, \{[a_2, b_2], I_2\}, \ldots)$$

Our step is to take the next element of the stack if the error estimate is less than the tolerance. Else, split into two new elements; that is, stack new elements at end and move on.
The class went on to talk about heaps, stacks, queues, and the differences thereof.

Remark: Recall that when we built `bisection.m`, we accepted (but did not use) a parameter p. We'll be using a parameter of Legandre polynomials (and we'll need to specify which one) to pass into `bisection.m`.

Lecture ends here.