2G03: Numbers and Functions

AJM

February 9, 2015

This document details the rudimentary lessons learned in C. It describes the following algorithms: PI.C, SINE.C, DIST.C, INTERP.C.

1 Estimating π

Euler proved two ways to compute π. The first is a straight summation:

$$\frac{\pi^2}{6} = \sum_{n=1}^{\infty} \frac{1}{n^2}, \tag{1}$$

but it converges very slowly. He then showed that the following recursion relation also works:

$$\frac{\pi}{2} = \sum_{n=1}^{\infty} t_n \tag{2}$$
$$t_n = t_{n-1}\frac{n-1}{2n-1}, t_1 = 1$$

and it converges much more rapidly. The recursion relation is truncated in PI.C when t_n satisfies:

$$t_n \leq \epsilon, \tag{3}$$

where ϵ is user supplied and sets the error in the estimate: if we set ϵ as 10^{-2} and 10^{-4} respectively, we get Δ values of 9.4359×10^{-3} and 1.1325×10^{-4} with minimum terms $t_7 = 5.3280 \times 10^{-3}$ and $t_{13} = 6.0588 \times 10^{-5}$. Note that PI.C is written in single precision so Δ can only be accurately estimated for $\epsilon \gtrsim 2 \times 10^{-7}$, with $t_{21} = 1.8553 \times 10^{-7}$; the smallest ϵ one may set is 10^{-45} ($t_{146} = 1.4013 \times 10^{-45}$). The program has also been modified to not accept any number less than 0 (in single precision this is equivalent to setting $\epsilon < 10^{-45}$).

2 Taylor Series Approximation for sin(x)

There are a number of series that return $\sin x$. One simple method is to use a Taylor series expansion about the value $x = a$:

$$f(x) = \sum_{i=0}^{\infty} \frac{\mathrm{d^i f(x)}}{\mathrm{dx^i}} \frac{(x-a)^i}{i!} \tag{4}$$

truncated at some term (again, the error in the estimate is set by the truncation term). In this program, the series is truncated in at the third term:

$$\sin x \approx x - \frac{1}{6}x^3 + \frac{1}{120}x^5 + O(x^7) \tag{5}$$

where we have expanded about $x = 0$. The following table represents the estimates for four values of x using the truncated Taylor expansion and that given by the sine function used in the standard C library $< math.h >$.

x	$< math.h >$	Taylor	Δ
0.5	0.479426	0.479427	0.000002
1.0	0.841471	0.841667	0.000196
2.0	0.909297	0.933333	0.024036
4.0	-0.756802	1.866667	2.623469

The Taylor series approximation is only good around the value a; in this case, it performs horribly for values greater than 2.0. Values greater than π also presents problems since the Taylor series does not map onto the Cartesion unit circle. In order to improve the estimate, the code has been modified to determine which quadrant x lies in[1], sets both a and the sign appropriately, and then evaluates the series. The resulting values are shown below.

x	$< math.h >$	Taylor	Δ
0.5	0.479426	0.479427	0.000002
1.0	0.841471	0.841667	0.000196
2.0	0.909297	0.909789	0.000491
4.0	-0.756802	-0.756872	0.000069

3 Vectors using Structures

The DIST program computes the distance between two vectors in $\Re^3$ space. The command line prompts for each vector component to be entered, separated by whitespace. Once both vectors $\mathbf{D}_1$ and $\mathbf{D}_2$ are entered, the program returns $||\mathbf{D}_1||$, $||\mathbf{D}_2||$, and $||\mathbf{D}_2 - \mathbf{D}_1||$. The novelty is that rather than passing variables, it uses a structure system.

4 Linear Interpolation

The simplest interpolation scheme between two points $[x_i, y_i]$ and $[x_{i+1}, y_{i+1}]$ is to use a linear fit to the points at x:

$$y \approx \frac{y_{i+1} - y_i}{x_{i+1} - x_i} x + x_i \tag{6}$$

The quality of the interpolation depends on the how well a linear function approximates the function $y(x)$ between the sets of points. The program uses LOOKUP.C to search for the points that bracket the user supplied value for x in order to calculate y. Since the points are sorted, the lookup function is a simple for loop combined with an if statement. It returns the location of the lower bracket, i, and passes it into INTERP.C. Using $f(x) = \frac{3}{4}x^2 + 5$ as a test function yields the following results.

x	$f(x)$	y
0.52	5.2028	0.3900
3.75	15.5469	22.6875
4.31	18.9321	33.0925
9.99	79.8501	151.3575
10.04	80.6012	168.1300

5 Numerical Calculus using Finite Differences

The mathematical definition of a derivative is:

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}, \tag{7}$$

[1]It also ensures that values greater than 2π are shifted to ensure periodicity.

while the integral is the limit of the Riemann sum:

$$\int_0^L f(x)\,\mathrm{d}x = \lim_{h\to 0}\sum_{n=1}^{N} f(x_n)h. \tag{8}$$

The method of finite differences uses these definitions to derive approximations — that is, h does not go to zero but is left at some finite value — whose error can be derived from the appropriate Taylor approximation.

5.1 Differentiation

The first order approximation to calculate the numerical derivative of the function $f(x)$ is given by:

$$\frac{\mathrm{d}f}{\mathrm{d}x} \approx \frac{f(x+\Delta x)-f(x)}{\Delta x} = \frac{\mathrm{d}f}{\mathrm{d}x} + \Delta x\frac{1}{2}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} + O(\Delta x^2) \tag{9}$$

The leading error term is of order Δx and scales as the second order derivative of the function. As test, we use the following function:

$$\begin{aligned} f(x) &= 1 - x + 4x^2 - x^3 \\ f'(x) &= -1 + 8x - 3x^2 \\ f''(x) &= 8 - 6x \end{aligned}$$

The root-mean-square of the difference in the integral at each point will be used to guage the error in the integral:

$$\sigma_{rms} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\frac{f(x+\Delta x)-f(x)}{\Delta x} - \frac{\mathrm{d}f}{\mathrm{d}x})^2} \tag{10}$$

From the above definition of the numerical derivative, σ_{rms} can be rewritten as:

$$\begin{aligned} \sigma_{rms} &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\Delta x\frac{1}{2}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} + O(\Delta x^2))^2} \\ &\approx \sqrt{\frac{\Delta x^2}{4n}\sum_{i=1}^{n}(\frac{\mathrm{d}^2 f}{\mathrm{d}x^2})^2} \\ &\propto \frac{1}{n} \end{aligned}$$

This table below shows the result of differentiating $f(x)$ over the interval [0,1].

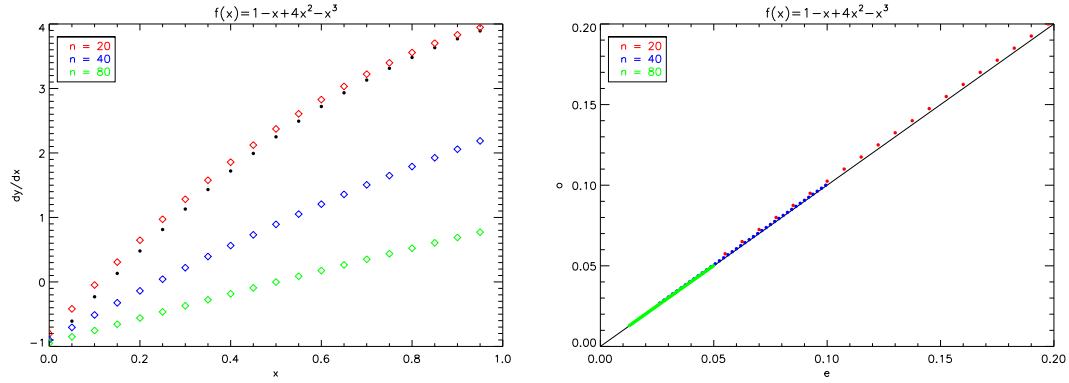
n	10	20	30	40	50	60	70	80	90	100
σ_{rms}	0.2692	0.1335	0.0887	0.0664	0.0531	0.0442	0.0379	0.0331	0.0295	0.0265

As expected, a least squares fit shows that σ_{rms} indeed goes as n^{-1}. In Figure **??** the error and rms scaling are shown.

5.2 Integration

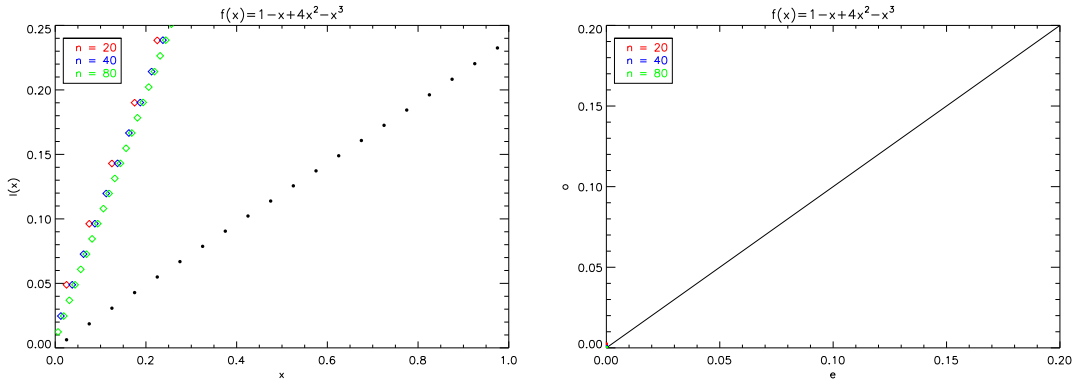
We can write the approximate integral of $f(x)$ as:

$$\int_0^L f(x)\,\mathrm{d}x \approx \sum_{n=1}^{N} f(x_n)\Delta x = \int_0^L f(x)\,\mathrm{d}x + \Delta x^2\frac{L}{8}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} + O(\Delta x^3) \tag{11}$$



(a) Differntial (b) Error

Figure 1: Numerical differentiation.



(a) Integral (b) Error

Figure 2: Numerical integration.

where each point x_n is given by $\Delta x(n - \frac{1}{2})$ — the midpoint rule. We can also write the rms error as:

$$\sigma_{rms} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\sum_{n=1}^{i} f(x_n)\Delta x - \int_0^{x_i} f(x)\,\mathrm{d}x)} \tag{12}$$