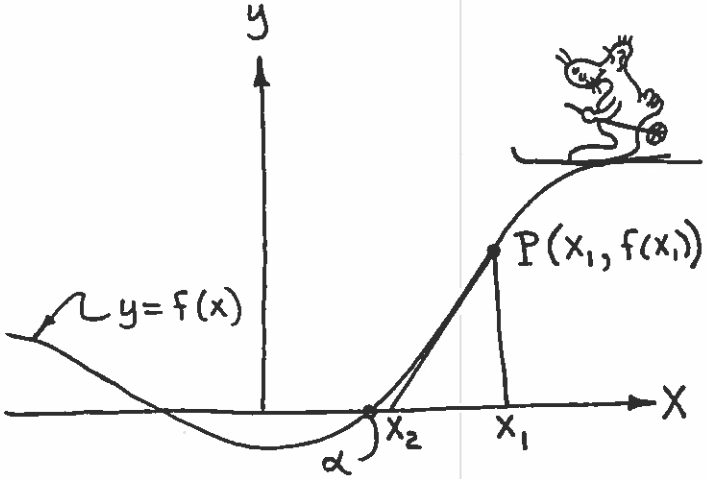
Issac Newton's Very Own Method for Solving Nonlinear Equations

Suppose you had a function $y = f(X)$ and you wanted to find its roots, i.e. places where $f(X) = 0$. In the picture, α is the unknown root we are looking for.



If our Intrepid Root Finder starts by guessing that the root lies at a point X_1, he will find that he isn't at the root at all, but is standing on a point P way up on the curve. However, notice that the <u>Tangent</u> to the curve at P cuts the X axis at a point X_2. Note also, that X_2 is much closer to α than was our starting guess X_1.

If we can calculate the <u>Slope</u> of the

Tangent at P, then we can calculate X_2. We could then repeat the process, using X_2 in place of X_1. Etcetera, etcetera, etcetera.

After a while, if we are on a nice, beginner's hill and not a bumpy expert's slalom course, we would find we had zoomed in pretty close to α. When we saw that our $f(X)$ was sufficiently close to zero, we could quit, rashly assuming we had found α.

Since the tangent at P is

$$y - f(X_1) = f'(X_1)\left[X - X_1\right]$$

it is obvious (as they say whenever anything is obscure) that

$$X_2 = X_1 - \frac{f(X_1)}{f'(X_1)}$$

If you use this as your basic iteration scheme, you'll zoom in like gangbusters, if the curve doesn't have any dips and bumps between you and the root. If the correction term has n zeros after the decimal point, then the result is generally good to about 2 n decimal places.

Incidentally, a good feature of the Newton method is that minor errors will correct themselves. This is nice if you are using it by hand.

> "Nothing in education is so astonishing as the amount of ignorance it accumulates in the form of inert facts!"
>
> *Henry Adams*

True. Therefore try the following homework. (Snicker, snicker!)

Write a FORTRAN program using Newton's method to find the roots of a function in the vicinity of some given starting guess. Your program should use two Function subprograms, one called FUNK and the other called DERIV. You will also be writing these two function subprograms. One will contain the function whose root is desired and the other will contain the derivative of that function.

Got that? O.K. Now use Newton's method to find the three roots of the polynomial

$$X^3 - 8X - 4 = 0$$

One root is near −2.; one is near 0.; and the last is between 2 and 4.5. Find the roots to within four decimal places of accuracy.

Verify each root that you find by

checking to see if it really does satisfy the polynomial. Have the computer calculate

$$R = X^3 - 8X - 4$$

R should equal zero for each root, but there will actually be some residue due to small round-off errors, truncation errors, and the fact that you quit before the computer died of exhaustion. The smaller the value of R, the better the root.

Check all three roots and print out both the root and the residue for each one.

Want to try something else? Write a program using Newton's method to determine square roots and cube roots of numbers. Prove your program on a list of at least 10 numbers, including 64, 10, 0.3, and 3.14159. Compare your results with those given by SQRT and ** (1./3.) Print out a table showing the number; its square root by Newton's method; SQRT's square root; the difference (in E format); the cube root by Newton; the computer's cube root; and their difference in E format.

How does Newton's Method have anything to do with square roots??

Hint: Let

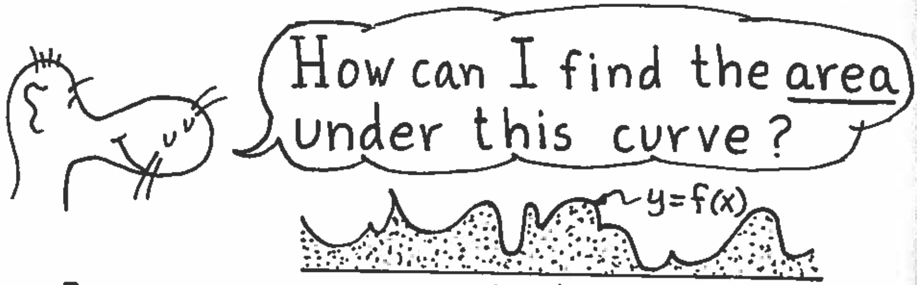
$$F(X) = A - X^2$$

where A is the number whose root you want.

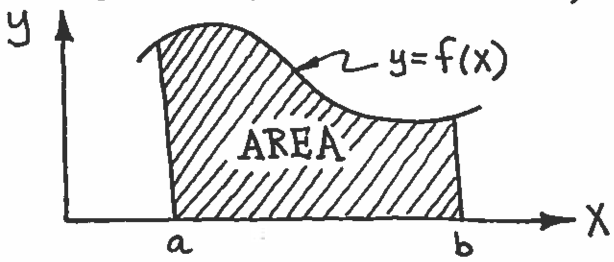
Added Little Brain Teaser:

The first sample program we saw (Pickles, Preserves, and Square Roots on Page 192) also calculated square roots. For $64,000.00 (or maybe some extra credit), is there any connection between that method for finding square roots and Newton's method for finding square roots? What is the connection? Is it French or via Peoria? Can you get there nonstop?

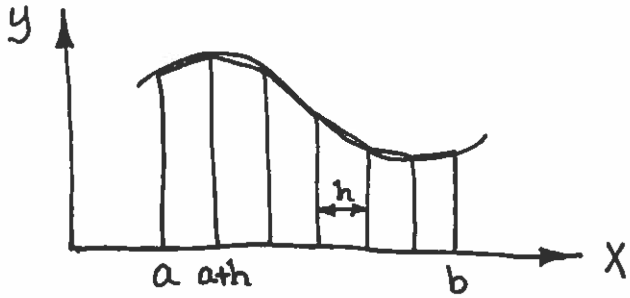
Incidentally, that other method for finding square roots is known as Mechanic's Rule. I suppose it was named after Ulysees S. Mechanic, the famous actor who for years played the role of Roy Rogers' horse Trigger.



An easy way to calculate the area under a curve $y = f(x)$, from $a \leq x \leq b$,



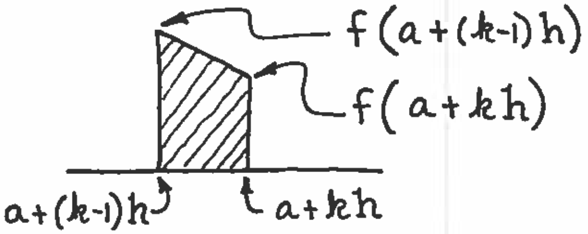
is to subdivide the area under the curve into vertical strips of equal width. The top of each strip can be approximated by a straight line as shown below:



Thus, the area under the curve is approximately the sum of the areas of n trapezoids of width h.

$$nh = b - a$$

The area under the k^{th} trapezoid



is given by:

$$\frac{h}{2}\left[f(a+(k-1)h) + f(a+kh)\right]$$

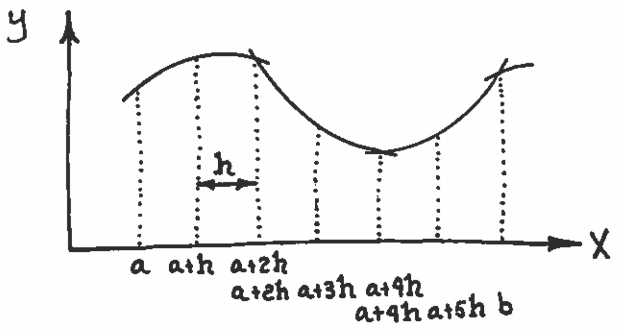
The total area under the curve from a to b is then approximately

$$A=h\left[\frac{f(a)+f(b)}{2}+f(a+h)+f(a+2h)+f(a+3h)\ldots+f(a+(n-1)h)\right]$$

Notice that at No Time did the word "Integration" pass my lips! You don't need to know calculus to find areas this way. The above procedure is obviously extremely easy to understand and to program.

Some people call the preceding formula the Trapezoidal Rule for Integration. Of course, by this name, it is much more difficult to grasp.

Another method for area estimation which is still easy but slightly more accurate than the Trapezoidal Rule involves replacing the straight line tops with parabolas:



You pick sets of 3 points off of the curve and then find the parabola that passes through them. (There is only one!) After hairy algebraic tedium, you find that the area under these parabolas is given exactly by a simple formula. If the points are evenly spaced a distance h apart, the area under the parabola from a to a+2h is given by

$$\frac{h}{3}[\,f(a)+4\,f(a+h)+f(a+2h)]$$

Similarly, the area under the parabola from a+2h to a+4h is:

$$\frac{h}{3}[\,f(a+2h)+4f(a+3h)+2f(a+4h)]$$

and so forth.

Thus, in toto, the area under the parabolas approximating the curve from a to b is:

$$\text{AREA}=\frac{h}{3}[f(a)+4f(a+h)+2f(a+2h)+4f(a+3h)+2f(a+4h)+\ldots+2f(a+(n-2)h)+4f(a+(n-1)h)+f(b)]$$

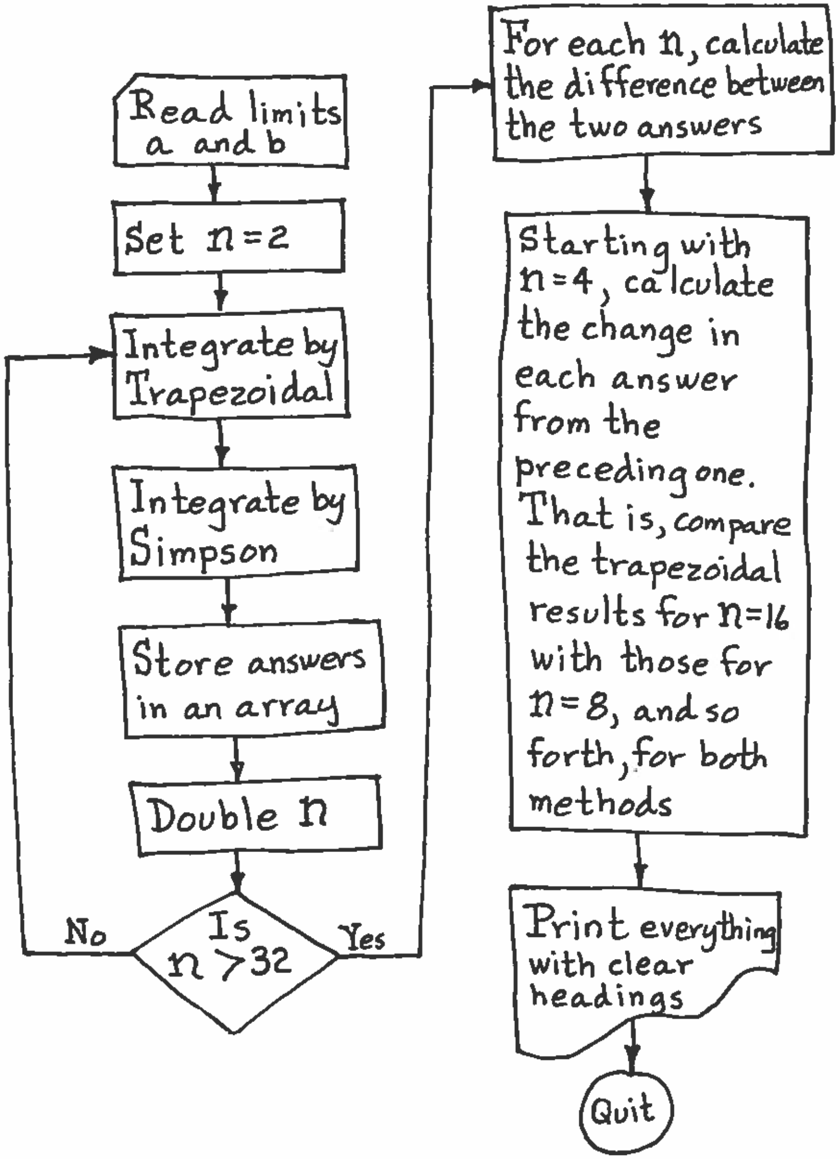
if there are n divisions. (Be sure n is even!) This formula is called <u>Simpson's One-Third Rule for Integration.</u>

This integration stuff is easy!

It really is, so try a few problems.

First, write a numerical integration function subprogram based on the Trapezoidal Rule. Once that works, write a second function subprogram based on Simpson's Rule. These two programs should each be able to integrate another function subprogram named "FUNK," using any desired number of divisions and any given starting and ending points a & b.

Next write a mainline program. This main program is going to compare the results of the trapezoidal rule with those from Simpson's rule. It also is going to see what effect n has on the accuracy of the results. Here's how your mainline is to work:



Read limits a and b
Set n=2
Integrate by Trapezoidal
Integrate by Simpson
Store answers in an array
Double n
Is n>32
No
Yes
For each n, calculate the difference between the two answers
Starting with n=4, calculate the change in each answer from the preceding one. That is, compare the trapezoidal results for n=16 with those for n=8, and so forth, for both methods
Print everything with clear headings
Quit

Now, test out your programs by seeing if you can integrate the following functions between the given limits. You can either write a separate FUNCT program for each one and run them one at a time, or else you can try using an External statement if you know what that is. I suggest running them one at a time for now.

Try these:

$y = \frac{1}{2+X}$ from $X=0$ to $X=2$

(The answer should be 0.69315)

$y = \frac{1}{1+X^2}$ from $X=0$ to $X=1$

(The answer should be $\frac{\pi}{4}$)

$y = e^{-\frac{X^2}{2}}$ from $X=0$ to $X=0.4$

(I haven't the foggiest idea what the right answer is but I wouldn't be surprised if it was 0.389 something!)

$y=(X^2 - \cos X)e^{-X}$ from $X=-1$ to $X=1$

(? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?)