

Lecture 2: Introduction Part II and Solving Equations

Instructor: Professor Amos Ron

Scribes: Yunpeng Li, Mark Cowlshaw

1 Numerical Solution to Quadratic Equations

Recall from last lecture that we wanted to find a numerical solution to a quadratic equation of the form

$$x^2 + bx = c. \quad (1)$$

One obvious method for solving the equation is to use the familiar quadratic formula:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 + 4c}}{2}. \quad (2)$$

Now, the formula does *not* provide us with a solution that is written in decimal form; to get such a solution we need to evaluate the above expression. To this end, since most computers perform additions, subtractions, multiplications, and even divisions very fast, we should focus on the need to take a squareroot. In short, the “explicit” solution for the quadratic equation actually reduces our problem to finding an algorithm for computing squareroots. Even if we have such an algorithm, we have to ask if using the quadratic formula is the most efficient way to solve this equation; in other words, whether the best algorithm for finding squareroots is superior to the best algorithm that solves the quadratic equation without using the formula. As we shall show now, we can extend the powerful square root algorithm we proposed in the last lecture so that it solves general quadratic equations, making the use of the quadratic formula (2) unnecessary (and, in fact, inefficient).

2 Finding Square Roots and Solving Quadratic Equations

2.1 Finding Square Roots

As we discussed last time, there is a simple scheme for approximating square roots to any given precision. More formally, we can find a nonnegative solution to the quadratic equation:

$$x^2 = c, x \geq 0, c \geq 0 \quad (3)$$

using an iterative method that allows us to control the precision of the solution. The idea is that we find a function that, given an approximation of the solution as input (x_{old}), outputs a more precise approximation (x_{new}). If we use this function iteratively by recycling the values it produces on each iteration, we will arrive at better and better solutions. We can continue this process until we reach any level of precision we want.

Recall that the iteration we used was:

$$x_{\text{new}} = \frac{x_{\text{old}}^2 + c}{2x_{\text{old}}} \quad (4)$$

An interesting property of this algorithm is that, roughly speaking, the precision of the output (x_{new}) is *doubled* with each iteration. As we shall see later, this speed of convergence of the algorithm will be an important factor in evaluating numeric algorithms.

2.2 Solving Quadratic Equations

Now that we have a scheme for solving a restricted kind of quadratic equation, can we use the scheme to solve our original problem? The answer is yes. To solve equations of the form

$$x^2 + bx = c \tag{5}$$

We simply need to add another term to the denominator of the formula:

$$x_{\text{new}} = \frac{x_{\text{old}}^2 + c}{2x_{\text{old}} + b} \tag{6}$$

We can use this new formula iteratively to arrive at numerical solutions of the quadratic equation that are arbitrarily precise. Each iteration will take only 5 operations (2 multiplications, 2 additions, and a division).

Conclusion: the explicit quadratic equation reduced our problem from iterating with (6) to iterating with (4). While a pedantic student may argue (correctly!) that the latter iteration involves fewer operations, we should all agree that the thought-to-be-wonderful quadratic formula has actually only a fringe value (at most).

So, it should now be clear that algorithms that are good for finding numerical solutions are often quite different from the methods we use to find analytic solutions.

3 Calculating Definite Integrals

Another good example of the difference between numerical computation and analytic solutions is in calculating definite integrals. Recall that the definition of the definite integral of a continuous (and we will assume positive) function $f(t)$ over the interval $[a, b]$, $\int_a^b f(t)dt$ is the area under the curve, as shown in Figure 3.

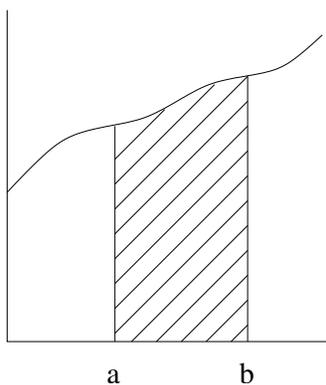


Figure 1: The Definite Integral of $f(t)$ over $[a, b]$

What is the best way to calculate the definite integral? Our experience in math classes suggests that we might use the anti-derivative approach. We find a function F such that $F' = f$, and then

$$\int_a^b f(t)dt = F(b) - F(a). \tag{7}$$

But this brings up two important questions:

1. How do we determine an anti-derivative function?
2. How difficult is it to evaluate the anti-derivative function at a and b ?

Sometimes the answer to both questions is “it’s easy”. For example, (a) consider $\int_0^1 t^3 dt$. We know how to compute an anti-derivative, namely $\frac{1}{4}t^4$, and since this anti-derivative is a polynomial, we know how to evaluate it efficiently using a few arithmetic operations. So we evaluate $\frac{1}{4}t^4|_0^1 = \frac{1}{4} - 0 = \frac{1}{4}$.

Unfortunately, most examples are not so nice. For example, (b) consider $\int_1^x \frac{1}{t} dt$. We know how to compute an antiderivative, namely $\log_e t$, so we can rewrite the integral as

$$\int_1^x \frac{1}{t} dt = \log_e x - \log_e 1 = \log_e x. \tag{8}$$

To get a numeric solution, we now need to evaluate \log_e at x . Unfortunately this is difficult - much more complicated than evaluating $\frac{1}{t}$. Any algorithm for evaluating \log will be more complicated than one for evaluating $\frac{1}{t}$, while the area that we are after is completely determined by $\frac{1}{t}$.

Some examples are even worse. (c) Consider $\int_a^b e^{-t^2} dt$, an integral that is useful in statistics. The integrand does have an anti-derivative, say $F'(t) = e^{-t^2}$. But it is not easy to compute the anti-derivative, or even to write down a simple closed-form. In fact, an efficient way for evaluating F at x is to compute the integral $\int_0^x f(t) dt$.

Finally, (d) consider the integral of the step function shown in Figure 3. Computing this integral

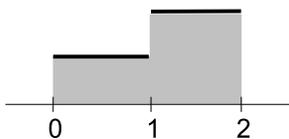


Figure 2: Approximating the Definite Integral Using Subintervals

is trivial. But no anti-derivative exists.

The above discussion highlights a fundamental issue: the area we need to compute depends on the function f . Our attempt to use the anti-derivative approach brings to the scene a new function F whose properties may not be in tandem with those of the original f . So is there a method for calculating the definite integral that uses only information about the function f ? The answer is yes, we can use the *definition* of the definite integral to approximate its value, as shown in Figure 3. We start by splitting up the interval into subintervals of equal size h , then estimating the definite

integral over each subinterval, then adding them all together. If we split the interval $[a, b]$ into N subintervals of equal width $h = \frac{b-a}{N}$, we can calculate the definite integral as:

$$\int_a^b f(t)dt = \sum_{j=1}^N \int_{a+(j-1)h}^{a+jh} f(t)dt. \quad (9)$$

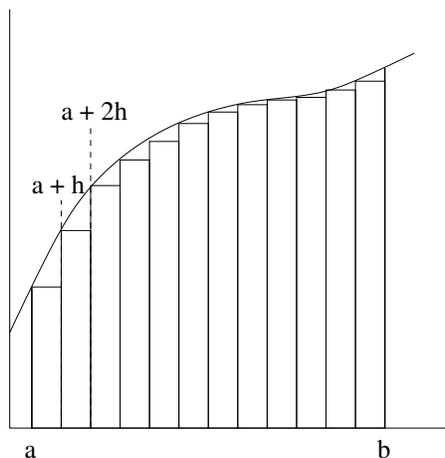


Figure 3: Approximating the Definite Integral Using Subintervals

We have reduced the problem to estimating the definite integral over smaller subintervals. Here are two possible strategies for estimating the smaller definite integrals:

Rectangle Rule

$$\int_a^{a+h} f(t)dt \simeq f(a)h \quad (10)$$

In the rectangle rule, we calculate the area of the rectangle with width h and height determined by evaluating the function at the left endpoint of the interval.

Midpoint Rule

$$\int_a^{a+h} f(t)dt \simeq f\left(a + \frac{h}{2}\right)h \quad (11)$$

In the midpoint rule, we calculate the area of the rectangle with width h and height determined by evaluating the function at the midpoint of the interval.

These two methods are depicted in Figure 3. Using these methods, we can calculate the definite integral by simply evaluating the original function at definite points, which we should always be able to do (the prevailing assumption is that we have an algorithm for evaluating f . In the absence of such algorithm, we actually do not know what f is). We can improve the precision in each case by changing the size of the subintervals h - the smaller the subinterval, the greater the precision.

The next question is, how can we decide which of these algorithms to use? Intuitively, it seems that the midpoint rule would be a better choice, but how can we quantify this intuition?

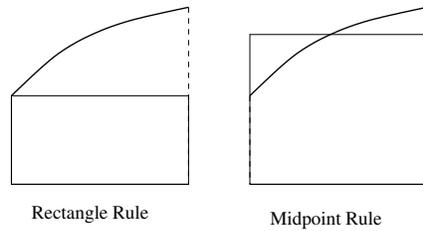


Figure 4: The Rectangle and Midpoint Rules

3.1 Evaluating Numeric Algorithms

In general, we evaluate numeric algorithms using 3 major criteria:

Cost - The amount of time (usually calculated as the number of operations) the computation will take. Frequently we can tradeoff cost versus accuracy by changing a parameter like the number of subintervals.

Accuracy - How quickly the algorithm approaches our desired precision. If we define the error as the difference between the actual answer and the answer returned by the numeric algorithm, we often measure the rate at which the error approaches 0 as cost is increased.

Robustness - How the correctness of the algorithm is affected by different types of functions and different types of inputs. The algorithm for calculating square roots turns out to be extremely robust: it will work for any square root, even if we give it a terrible initial value.

3.2 Evaluation of Algorithms for Definite Integrals

Applying these criteria to the midpoint and rectangle rules:

Cost Each algorithm performs a single function evaluation to estimate the definite integral over a subinterval, so the cost is identical. The total cost over the entire computation grows linearly with the number of intervals N in each case, meaning that there is some fixed constant c_1 , not dependent on the number of intervals, such that the total cost is $c_1N = O(N)$. For example, if the number of intervals doubles, the cost will double as well. Note that we can tradeoff between cost and accuracy by changing the number of intervals N .

Accuracy Clearly as the size of each subinterval h gets smaller, the error gets smaller as well. Is there any difference between the rate at which the rectangle and midpoint rules' error approaches 0?

Rectangle Rule - For the rectangle rule, the error approaches 0 in direct proportion to the rate at which h approaches 0. In other words, the error $\epsilon = c_1h$ for some fixed constant c_1 , or $\epsilon = O(h)$. For example, halving the size of h will halve the amount of error.

Midpoint Rule - For the midpoint rule, the error approaches 0 in proportion to the rate at which the *square* of h approaches 0. That is the error $\epsilon = c_2h^2$ for some fixed constant c_2 , or $\epsilon = O(h^2)$. For example, halving the size of h will divide the error by 4.

This difference is substantial. For example, if we want to compute the integral of a function between 0 and 1, and we use $N = 1000$ subintervals, each subinterval will have size $h = \frac{1}{1000}$, so the rectangle rule's error will be on the order of $\frac{1}{1000}$, while the midpoint rule's error will be on the order of 10^{-6} .

Robustness It turns out that for functions that are reasonably well-behaved, both the endpoint rule and the midpoint rule are quite robust. There do exist situations, however, in which it is possible to evaluate a function at the left endpoint of an interval, and not at the midpoint.

Through this evaluation, we have shown that, in general, the midpoint rule is a *far* better choice than the rectangle rule for approximating the definite integral, since, for the same cost, it will give us much more precision.

4 Loss of Significance

One final way that numerical computation is different than the way you have done mathematics to this point is the potential loss of significance. Consider two real numbers stored with 10 digits of precision beyond the decimal point:

$$\begin{aligned}\pi &= 3.1415926535 \\ b &= 3.1415926571\end{aligned}$$

The actual numbers π and b contain additional digits that we have not stored. So our stored numbers are actually good, acceptable approximations of the true π, b . Now, we want to compute $\pi - b$, and want to have a similarly good approximate representation: 10-11 significant digits, i.e., once the zeros end and the number begins. However, all we can do is subtract the given approximation to obtain 0.000000036: only two significant digits! So, we have lost most of the precision of the two original numbers. One might try to solve this problem by increasing the precision of the original numbers, but this is not a solution: For any finite precision storage, numbers that are close enough will be indistinguishable.

There is no universal way to avoid loss of precision! The only general solution is to avoid subtracting numbers that are almost equal.

An example of the effect of loss of significance occurs in calculating derivatives.

EXAMPLE 4.1 (Calculating Derivatives). *Given the function $f(t) = \sin t$, what is $f'(2)$?*

We all know from calculus that the derivative of $\sin t$ is $\cos t$, so one could calculate $\cos(2)$ in this case. However, just as with calculating definite integrals, it is not always possible, or efficient, to evaluate the derivative of a function.

Instead, recall that the value of the derivative of a function f at a fixed point t_0 is the slope of the line tangent to f at t_0 . Recall, also, that we can approximate this slope by calculating the slope of a secant line that intersects f at t_0 and at a point very close to t_0 , say $t_0 + h$, as shown in Figure 4. We learned in calculus, that as $h \rightarrow 0$, the slope of the secant line approaches the slope of the tangent line, or:

$$f'(t_0) = \lim_{h \rightarrow 0} \frac{f(t_0 + h) - f(t_0)}{h}. \quad (12)$$

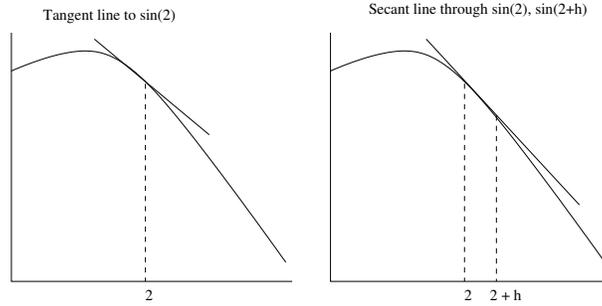


Figure 5: Tangent and Secant Lines on $f(t) = \sin t$

Just as with the definite integral, we could approximate the derivative at t_0 by performing this calculation for small values of h . However, notice that as h gets very small, the numerator of Equation 12 will become 0, due to loss of significance, so that we will erroneously calculate all derivatives as 0.

Preventing loss of significance in our calculations will be an important part of this course.

5 Topic I: Solving Equations

Now that we have some idea of what algorithms for computation are, we will discuss algorithms for performing one of the most basic numerical tasks, solving equations in one variable.

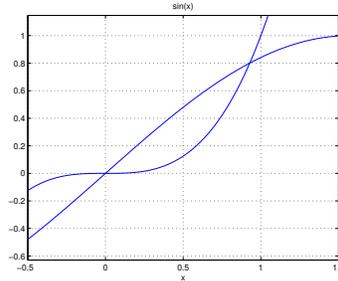


Figure 6: Graphical Solution for $x^3 = \sin x$

Consider the following equation, with solution depicted in Figure 5.

$$x^3 = \sin x. \quad (13)$$

There is no analytic solution to this equation. Instead, we will present an iterative method for finding a numerical solution.

5.1 Iterative Method

As in our solution to finding square roots, we would like to find a function g , such that if we input an initial guess at a solution, g will output a better approximation of the actual solution. More

formally, we would like to define a sequence x_0, x_1, x_2, \dots with $x_{n+1} \leftarrow g(x_n)$ such that $(x_n)_0^\infty$ converges to the solution. We can continue calculating values x_i until we reach two values with $x_j = x_{j+1}$.

Clearly, one property of g is that $g(r) = r$, where r is the real solution to Equation 13. A transformation of Equation 13 (e.g. $x = \sin x - x^3 + x$) will satisfy this property, the trick is to choose the transformation that produces a converging series of values x_1, x_2, \dots .