Introduction to Numerical Analysis

Scope and purpose of NA; examples

This is what Lloyd Trefethen wrote some time back:

> *Numerical analysis is the study of algorithms for the problems of continuous mathematics.*

Algorithms are step-by-step methods for doing a calculation; the **problems of continuous mathematics** include finding solutions (in real or complex numbers) to equations, evaluating definite integrals, solving differential equations, finding optimal points on continuous functions.

Examples:

- find x such that $e^{-x/7}(x^2/2 - x/6) = .001$
- find a function $u(s)$ which satisfies $u'' - su\sin(2u) = 0, u(0) = 1, u(1) = 0$
- Solve $Ax = b$ where A is very large, sparse matrix.
- ... and many more.

For examples of algorithms, see the Bisection Method and Newton's Method in the section on Solving Equations in One Dimension.

Errors are a way of life in scientific computing

All computer calculations of actual numbers, except exact integer arithmetic, gives results at best approximately correct. This is because they use finite precision arithmetic: almost every calculation involves rounding to the nearest available number.

For example, if you use only three decimal places, the best you can do to represent 1/3 is .333, which is not completely accurate.

To discuss this mathematically, we need the concept of **error**. Note that error may also arise from the numerical method itself; such errors are often called **trunction errors** for reasons that will become clear later. Errors due to finite precision arithmetic are called **rounding errors**.

Here are the definitions we use on this course (note: mathematics without definitions is extremely rare; definitions may differ from course to course, from book to book—even from lecture to lecture, but I hope not on this course).

Error: if we have a quantity whose true value is x and its approximate value is $\tilde{x}$, then the error e in the approximation is given by $e = \tilde{x} - x$. In the example above, the error is $0.333 - \frac{1}{3}$, which is $-0.00033333...$.

Absolute error $|e|$ follows from the above: $|e| = |\tilde{x} - x|$.

Relative error is $r = |\tilde{x} - x|/|x|$. In the example above, $r = (0.333 - \frac{1}{3})/(1/3) = 0.001$.

The symbols r and e above are for convenience, and many others are also in use—sometimes we even use no symbol at all, but just calculate the required value directly.

Algorithms are about reducing error. We study how well they do that, in the sense of *how reliably* they reduce *which* error *how fast*. Good algorithms always send the error to zero very quickly (and it may be that in this sense *no* good algorithms exist).

Finite precision arithmetic

There are many ways to represent numbers on a computer, but one above all stands out: floating point numbers.

A floating point number f has a **mantissa** and an **exponent**. The mantissa length determines the maximum number of siginificant figures which can be correct, and the exponent determines the size of the number.

The definition above is general. On this course, we use the following idealised definition: a floating point number f is defined by $f = 0$ or by $f = \pm\frac{m}{\beta^t} \times \beta^e$, where β is the base, m/β^t is the mantissa and e is the exponent.

The expression $F(\beta, t, L, U)$ to denotes the following set of numbers: the number zero and the floating point numbers with mantissas $\pm m/\beta^t$, where $m \in \{1, 2, 3, \ldots, \beta^t\}$ and exponents $e \in [L, U]$. Informally, these are the base-β floating point numbers with at most t significant digits.

A common example is $F(2, 53, -1023, 1024)$. This set is part of the IEEE double precision standard, which therefore specifies binary numbers with 53 bits in the mantissa and 2048 different exponents.

We will often think about sets like $F(10, 4, -20, 20)$ (four-place decimal numbers) or $F(2, 3, 0, 2)$ (three-place binary numbers). Lets write out the positive elements of the latter set completely:

For $e = 0$: $\{1/8, 2/8, 3/8, 4/8, 5/8, 6/8, 7/8, 8/8\}$

For $e = 1$: $\{1/4, 2/4, 3/4, 4/4, 5/4, 6/4, 7/4, 8/4\}$

For $e = 2$: $\{1/2, 2/2, 3/2, 4/2, 5/2, 6/2, 7/2, 8/2\}$

Important properties of $F(\beta, t, L, U)$:

- Largest positive element is β^U (since the largest mantissa is 1).
- Smallest positive element is β^{L-t} (since the smallest mantissa is β^{-t}).
- There are $\beta^t(U - L + 1)$ positive elements in F, and hence $1 + 2(\beta^t(U - L + 1))$ elements in all.

- Some elements in F may represent equal real numbers. For example, if $\beta = 2$ and $m = 2k$ then $m/2^t \times 2^e = k/2^t \times 2^{e+1}$ (this doesn't apply when $e = U$, of course). Therefore half the numbers in a base 2 system that are represented with exponent $e + 1$ are also represented with exponent e (for all but the largest e, of course). This means that the real numbers representable in a base 2 system (by our definition of $F(\beta, t, L, U)$, of course) are approximately half as numerous as the elements of the system.
- F represents real numbers that are uniformly spread between β^{e-t} and β^{e+1-t}. For example, the set $F(2, 3, 0, 2)$ written out above contains the sequence of values $4/4$, $5/4$, ..., $8/4$ which gives 5 numbers in the interval 1 to 2 separated by $1/4$ from each their neighbours. Note that only about half of the available mantissas are used to fill the space between two powers of β—another illustration that F represents fewer numbers than it has elements.

We define the function fl : $\mathbb{R} \to F$ by fl$(x) =$ element in F nearest to x.

Example: if $F(2, 5, -9, 9)$, then fl$(2/3) = 21/32 \times 2^0$ (because $2/3$ of 32 rounds to 21); also fl$(\pi) = 25/32 \times 2^2$.

Theorem: the relative error in fl$(x) \approx x$ is at most $\frac{1}{2}\beta^{1-t}$. (Proof in class notes).

(Some exercises here would be a good idea. Submissions are welcome.)

Solving equations in one dimension

That is, given $f(x) = 0$, find x. We only study the case where $f : \mathbb{R} \longrightarrow \mathbb{R}$ (that is, f is a function mapping real numbers to real numbers). This is trivial for linear equations, which have the form $ax = b$ and solution $x = b/a$. Linear equations are therefore only interesting in higher dimensions (and numerical linear algebra is an important part of 2NA in MAM246W).

So we are only concerned with the following problem: for a nonlinear function f on real numbers, find (some or all) x such that $f(x) = 0$.

This is hard: there is no guaranteed method. Indeed, there is no way of knowing in general how many solutions there are, if any.

We will consider some special cases, and several algorithms.

Fixed point iteration

ALGORITHM:

```
In: x0, function g, tol

1. set x = x0, g=g(x)

2. Iterate:  while  |x-g|>tol
                set x=g, g=g(x)

Out: x, g OR error message
```

Here, we test for convergence in the $x_k - g(x_k)$-values. It may be that these get smaller than `tol` while the corresponding f values are still unacceptably large. That is, closeness to the root may be defined by small $|f|$ rather than small $|x_k - x^*|$. Some real-world applications are like that, and in such cases one specifies a second tolerance to be used on $|f|$ and you only accept roots that satisfy both tolerances.

THEORY:

We write f in the form $f(x) = x - g(x)$ (that is, we find a g such that the roots of $x = g(x)$ are also roots of f). We then find a starting x_0 and iterate

$$x_{k+1} = g(x_k) \tag{1}$$

.

Finding good g and good x_0 may require creative thought.

Graphically, fixed point iteration amounts to drawing the graphs $y = x$ and $y = g(x)$ and then drawing the correct cobweb diagram (you can make sure your cobweb is correct by starting at the point $x_0, g(x_0)$).

Experiments: take an f, invent a variety of possible g, and test graphically for which of them the iteration goes towards the root. You should be led to a guess confirmed by the following:

Theorem: If $f(x^*) = 0$ and $x, x^* \in$ an iterval I such that $f(x) = x - g(x)$ and $|g'(x)| < C < 1$ on I, and $e_k = x_k - x^*$ is the error in the k-th approximation, where $x_0 \in I$ and x_k is calculated by equation 1, then fixed point iteration using g converges to the root, and

$$|e_{k+1}| < C|e_k|$$

Proof: given in lectures.

Note that $|g'(x)| < 1$ *near* the root, not just at the root.

The rate of convergence is *linear*, because e_{k+1} is bounded by a linear function of e_k. Such convergence is normally too slow for general use.

Bisection

Another linearly converging algorithm. It is often part of root-finding routines because, if it can start, convergence is guaranteed.

ALGORITHM:

```
In: function f, xL < xR s.t. f(xL)*f(xR) < 1, tol>0

1. Set error = 0.5*(xR-xL), set fL = f(xL), fR = f(xR)

2. Iterate: while error > tol
            set xM = (xL + xR)/2, fM = f(xM)
            if fM = 0
               set x = xM, break
            elseif fM*fR > 0
               set xR = xM, fR = fM
            else
               set xL = xM, fL = fM

Out: set x = xM, f=fM
```

In practice, one finds the starting `xL, xR` by experiment—either by guessing values or by plotting the graph of f.

THEORY

Only applies to continuous functions $f(x)$ for which x_0, x_1 are available such that $f(x_0)f(x_1) < 0$. This is known at the start, so bisection, when it can start, always works. Proof depends on the *intermediate value theorem*.

The rate of convergence is *linear* with convergence constant one-half: $|e_{k+1}| = 0.5|e_k|$. Proof (see lectures) follows directly from $e_0 = (x_R - x_L)/2$.

Corrolary: $e_k = (0.5)^k e_0$, so that the minimum n needed to meet a given error bound $e_n \leq$ `tol` is $n = 1 + \left\lceil \dfrac{\log(xR - xL)}{\log 2} \right\rceil$, where the brackets indicate the `ceil` function: the smallest integer upper bound.

Newton's method

Now we get a quadratically converging method, known as *Newton's method* (=NM) and also as *the Newton-Raphson method*. It has the advantage of rapid convergence when it works, but alas also the disadvantage that for it to work, lots of conditions have to apply (luckily, these conditions can often be met in practice—at least for one-dimensional problems).

Newton's method will also serve to introduce the technique of asymptotic expansion. This technique is fundamental to numerical analysis (and also cosmology, fluid dynamics, functional analysis and much else).

ALGORITHM

THEORY

Does not always work, and in general it cannot be known before you start whether it will work. In order to work, initial x_0 must be sufficiently close to a root. In practice, "sufficiently close" cannot be known in advance either.

When it works, and it is converging to a *simple* root, the rate of convergence is *quadratic*.

Proof of this theory takes several sections. First we show that NM considered as a fixed point iteration must always have an interval of convergence, provided that f''/f' is bounded over an interval containing the root. We aren't happy with this result, as it doesn't prove quadratic convergence, nor does it offer any help for the case $f'(x^*) = 0$. We then do the classical asymptotic analysis with which one proves quadratic convergence. This is good for you, because we use Taylor series for our asymptotic analysis, and Taylor series turn up all the time in phyics, chemistry and engineering (and these days even in biology and economics). However, the asymptotic approach is algebraically demanding, and so the overall pattern of the proof disappears and with it, all hope of insight into NM. So to summarise the result, we redo the proof much more simply.

Taylor polynomial expansion

Notes for rewriting: (1) This is a relatively straightforward application of derivatives. (2) Considered as asymptotic expansions, Taylor series have $O(h^n)$ properties. (3) Asymptotic expansions with the

$O(h^n)$ properties are a fundamental tool in numerical analysis.

The basis of much numerical analysis is the *Taylor polynomial expansion*:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \dots$$

This is a remarkable result. It means that the value of a function $f(x)$ at some point x and *all* its derivatives completely determines the value of the function a small distance h from x. Any differentiable $f(x)$ can be expanded in this way.

Heuristically (=gives a reasonable explanation which isn't quite a proof), we assume that $f(x+h) = \sum_{k=0}^{\infty} a_k h^k$. Set $h = 0$ to get a_0, take the derivative and set $h = 0$ to get a_1 and in general, take the k-th derivate and set $h = 0$ to get a_k.

First example: suppose $f(x)$ is the exponential function e^x. Then

$$e^{x+h} = e^x + he^x + \frac{h^2}{2!}e^x + \frac{h^3}{3!}e^x + \dots$$

since e^x is equal to all its derivatives. Putting $x = 0$ in this expansion gives the familiar (?) power series expansion for the exponential:

$$e^h = 1 + h + \frac{h^2}{2!} + \frac{h^3}{3!} + \dots \tag{2}$$

since $e^0 = 1$. (If the notation looks unfamiliar simply replace h by x in Equation (2) since both are just "placeholders".)

If h is "small", i.e. $h \ll 1$, we can write Equation (2) as

$$e^h = 1 + h + O(h^2).$$

The term $O(h^2)$ stands for "order of magnitude of h^2". It means that the largest term in the rest of the series has this order of magnitude (because $h \ll 1$ implies that $h^2 \gg h^3 \gg h^4$, etc.). This incidentally shows why we can approximate the exponential e^x to $1 + x$ when x is small.

As an exercise you should use the Taylor expansion to show that

$$\begin{aligned} \cos h &= 1 - \frac{h^2}{2!} + \frac{h^4}{4!} - \dots \\ \sin h &= h - \frac{h^3}{3!} + \frac{h^5}{5!} - \dots \end{aligned}$$

Derivation of Newton's method from Taylor's expansion

We want an iterative (repetitive) numerical method for solving the equation

$$f(x) = 0 \tag{3}$$

for a root x. The idea is to guess a starting value for x and to refine this guess repeatedly. Suppose the nth such refinement (called an *iterate*) is x_n. Ideally we would like to be able to find a value h so that the next iterate x_{n+1}, where

$$x_{n+1} = x_n + h, \tag{4}$$

hits the root exactly, i.e.

$$f(x_{n+1}) = 0, \tag{5}$$

i.e.

$$f(x_n + h) = 0, \tag{6}$$

substituting from Equation (4). Taylor-expanding the LHS of Equation (6) gives

$$f(x_n) + hf'(x_n) + \frac{h^2}{2!}f''(x) + \ldots = 0. \tag{7}$$

If we neglect all terms of $O(h^2)$ and smaller, Equation (7) becomes

$$f(x_n) + hf'(x_n) = 0,$$

which if we solve for h gives

$$h = -f(x_n)/f'(x_n). \tag{8}$$

Finally, substituting h from Equation (8) into Equation (4) gives

$$x_{n+1} = x_n - f(x_n)/f'(x_n), \tag{9}$$

which we (should) recognize as Newton's method for solving $f(x) = 0$.

Error analysis of Newton's method: proof of rate of convergence

Let α be a root of Equation (3), i.e. $f(\alpha) = 0$. Define the error after n iterations as ϵ_n, i.e.

$$x_n = \alpha + \epsilon_n \tag{10}$$

(clearly, it follows that $x_{n+1} = \alpha + \epsilon_{n+1}$.) The aim of much of numerical error analysis is to establish a relationship between ϵ_n and ϵ_{n+1} (i.e. the error after $n + 1$ iterations).

Recall, for example, that we showed for the direct iteration method that $\epsilon_{n+1} \propto \epsilon_n$. In the case of the Bisection method we saw that $E_{n+1} = \frac{1}{2}E_n$, where E_n is the worst possible error after n bisections. Both these methods are therefore referred to as having *linear* or *first-order* convergence.

Going back to Newton's method, we will now use some standard techniques of numerical analysis to show that it has *quadratic* or *second-order* convergence. Substituting Equation (10) into Equation (9) gives

$$\alpha + \epsilon_{n+1} = \alpha + \epsilon_n - \frac{f(\alpha + \epsilon_n)}{f'(\alpha + \epsilon_n)},$$

i.e.

$$\epsilon_{n+1} = \frac{\epsilon_n f'(\alpha + \epsilon_n) - f(\alpha + \epsilon_n)}{f'(\alpha + \epsilon_n)}.$$

Taylor-expanding f' and f in the numerator of the RHS gives

$$\epsilon_{n+1} = \frac{\epsilon_n f'(\alpha) + \epsilon_n^2 f''(\alpha) + \cdots - f(\alpha) - \epsilon_n f'(\alpha) - \frac{1}{2}\epsilon_n^2 f''(\alpha) - \cdots}{f'(\alpha + \epsilon_n)}.$$

After handy cancellations in the numerator of the RHS, and remembering that $f(\alpha) = 0$ (definition of the root α), we now Taylor-expand the denominator of the RHS to get

$$\epsilon_{n+1} = \frac{\frac{1}{2}\epsilon_n^2 f''(\alpha) + O(\epsilon_n^3)}{f'(\alpha) + \epsilon_n f''(\alpha) + \frac{1}{2}\epsilon_n^2 f'''(\alpha) + \cdots}, \tag{11}$$

where the term $O(\epsilon_n^3)$ stands for a collection of terms in ϵ_n, the *largest* of which has the order of magnitude of ϵ_n^3.

Now we would like to express the RHS of Equation (11) as a polynomial in ϵ_n, so we can see more easily how it relates to ϵ_{n+1}. This is not as difficult as it might seem, and is in fact part of the standard armoury of the numerical analyst. We begin by rearranging the denominator of Equation (11) so that the equation becomes

$$\epsilon_{n+1} = \frac{[\frac{1}{2}\epsilon_n^2 f''(\alpha) + O(\epsilon_n^3)]}{f'(\alpha)} \left[\frac{1}{1 + \frac{\epsilon_n f''(\alpha)}{f'(\alpha)} + \frac{\epsilon_n^2 f'''(\alpha)}{2f'(\alpha)} + \cdots} \right] \tag{12}$$

Note, incidentally, that we can divide safely by $f'(\alpha)$ in Equation (12) since we are not (at this stage) considering equal roots at α (so $f'(\alpha) \neq 0$).

To simplify Equation (12) we note first that

$$\frac{1}{1+r} = 1 - r + r^2 - r^3 + \cdots \tag{13}$$

(sum of a geometric series with a constant term of 1, or alternatively, the binomial expansion of $(1+r)^{-1}$). Now if we identify r in Equation (13) with the terms

$$\frac{\epsilon_n f''(\alpha)}{f'(\alpha)} + \frac{\epsilon_n^2 f'''(\alpha)}{2f'(\alpha)} + \cdots$$

after the 1 in the denominator of the expression in the square brackets on the RHS of Equation (12), we can rewrite Equation (12) as

$$\epsilon_{n+1} = \frac{[\frac{1}{2}\epsilon_n^2 f''(\alpha) + O(\epsilon_n^3)]}{f'(\alpha)} \left[1 - \left(\frac{\epsilon_n f''(\alpha)}{f'(\alpha)} + \frac{\epsilon_n^2 f'''(\alpha)}{2f'(\alpha)} + \cdots \right) + \left(\frac{\epsilon_n f''(\alpha)}{f'(\alpha)} + \frac{\epsilon_n^2 f'''(\alpha)}{2f'(\alpha)} + \cdots \right)^2 - \cdots \right] \tag{14}$$

Since the largest term after the 1 in the expression in the square brackets on the RHS of Equation (14) is $O(\epsilon_n)$, we can rewrite the equation as

$$\epsilon_{n+1} = \frac{\epsilon_n^2 f''(\alpha)}{2f'(\alpha)} + O(\epsilon_n^3),$$

i.e.

$$\epsilon_{n+1} = K\epsilon_n^2 + O(\epsilon_n^3), \tag{15}$$

where K is a constant depending on α. Since the RHS of Equation (15) does not contain a linear term in the error ϵ_n, the convergence in Newton's method is said to be *quadratic* or *second-order*. This explains analytically what we suspected numerically: that Newton's method is generally much faster than both the Bisection method and the method of direct iteration.

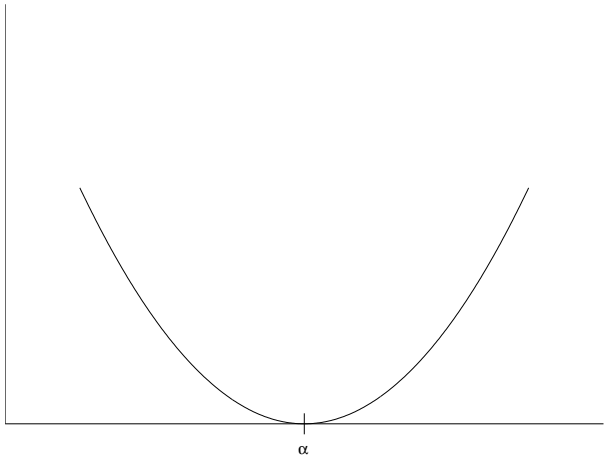

Figure 1: Two equal roots at $x = \alpha$

Theorem on Newton's method

The insights on NM from the preceding analysis can be summarised as follows: in order for the required $O(h^n)$ properties to hold, we need $|f'| > 0$ and $|f''| < \infty$ on an interval containing the root. It turns out that we can quantify the size of the interval in terms of the bounds $|f'| > \delta$ and $|f''| < \alpha$ on the interval.

The theorem below uses α via the mean value theorem for f'', which says that for $x, y \in I$ on which f'' exists we have $f'(x) - f'(y) = f''(\eta)(x - y)$ for some η in the interval between x and y. Clearly $|f''(\eta)| < \alpha$, so the MVT implies that $|f'(x) - f'(y)| < \alpha|x - y|$. (This use of MVT to bound derivatives is often called Lipschitz continuity.)

The theorem below establishes a relationship between $x_k - x^*$ and $x_{k+1} - x^*$ via the formula for Newton's method and the relationships mentioned above. It is very compactly presented, via integrals, but the ideas are fairly simple. You should study the theorem until you understand it fully, as the techniques in its proof are very common in all fields of mathematical analysis.

(van Loan's version the theorem goes here; see lectures)

The case of multiple roots in Newton's method

We have seen numerically that convergence in Newton's method slows down dramatically in the case of multiple roots. This is in fact a general result, which can be shown quite easily from the above analysis.

Let us consider the case of two equal roots at $x = \alpha$ (Figure (1). Then $f(\alpha) = 0$ as before, but in addition $f'(\alpha) = 0$ (but $f''(\alpha) \neq 0$). If we put $f'(\alpha) = 0$ in Equation (11) and rework the analysis from that point (but now taking the factor $f''(\alpha)$ out of the denominator instead of $f'(\alpha)$) we will find after not *too* much effort that

$$\epsilon_{n+1} = \frac{1}{2}\epsilon_n + O(\epsilon_n^2),$$

i.e. convergence is now *linear* as opposed to quadratic.

However, what is very neat is that changing Newton's formula to

$$x_{n+1} = x_n - 2f(x_n)/f'(x_n)$$

restores the more rapid quadratic convergence. Again, you can see this fairly easily by simply reworking the above analysis with the factor 2 in place and with $f'(\alpha) = 0$.

This result can be generalized as follows: if the equation $f(x) = 0$ has k roots which are all equal (i.e. the root has a multiplicity of k), the quadratic convergence of Newton's method is restored with the formula

$$x_{n+1} = x_n - kf(x_n)/f'(x_n).$$

This result can be established using the same approach as above.

Secant method, regula falsi?

These are methods that combine some features of the preceding methods. They illustrate various trade-offs between the guarantee of convergence, the speed of convergence and the complexity of the method.