Numerical Analysis and Computation

Algorithms and Convergence; Solutions of Equations of One Variable

Lecture Notes #2.5

Joe Mahaffy
Department of Mathematics and Statistics
San Diego State University
San Diego, CA 92182-7720
mahaffy@math.sdsu.edu
http://www-rohan.sdsu.edu/~jmahaffy

$Id: lecture.tex,v 1.25 2008/09/08 19:49:27 mahaffy Exp $

Definition: ***Algorithm —***
An **algorithm** is a procedure that describes, in an ***unambiguous manner***, a finite sequence of steps to be performed in a specific order.

In this class, the objective of an algorithm is to implement a procedure to solve a problem or approximate a solution to a problem.

Most homes have a collection of algorithms in printed form — we tend to call them "recipes."

There is a collection of algorithms "out there" called ***Numerical Recipes***, google for it!

Definition: ***Pseudo-code —***
Pseudo-code is an algorithm description which specifies the input/output formats.

Note that pseudo-code is ***not*** computer language specific, but should be easily translatable to any procedural computer language.

Examples of Pseudo-code statements:

for i = 1,2,...,n

Set $x_i = a_i + i * h$

While $i < N$ **do Steps 17 - 21**

If ... then ... else

Definition: ***Stability —***
An algorithm is said to be stable if small changes in the input, generates small changes in the output.

At some point we need to quantify what "small" means!

If an algorithm is stable for a certain ***range*** of initial data, then is it said to be ***conditionally stable***.

Stability issues are discussed in great detail in ***Math 543***.

Suppose $E_0 > 0$ denotes the initial error, and E_n represents the error after n operations.

If $E_n \approx \mathcal{C} E_0 \cdot n$ (for a constant $\mathcal{C}$ which is independent of n), then the growth is ***linear***.

If $E_n \approx \mathcal{C}^n E_0$, $\mathcal{C} > 1$, then the growth is ***exponential*** — in this case the error will dominate very fast (undesirable scenario).

Linear error growth is usually unavoidable, and in the case where $\mathcal{C}$ and E_0 are small the results are generally acceptable. — **Stable algorithm**.

Exponential error growth is unacceptable. Regardless of the size of E_0 the error grows rapidly. — **Unstable algorithm**.

The recursive equation

$$p_n = \frac{10}{3} p_{n-1} - p_{n-2}, \quad n = 2, 3, \ldots, \infty$$

has the exact solution

$$p_n = c_1 \left(\frac{1}{3}\right)^n + c_2 3^n$$

for any constants c_1 and c_2. (Determined by starting values.)

In particular, if $p_0 = 1$ and $p_1 = \frac{1}{3}$, we get $c_1 = 1$ and $c_2 = 0$, so $p_n = \left(\frac{1}{3}\right)^n$ for all n.

Now, consider what happens in 5-digit rounding arithmetic...

Now, consider what happens in 5-digit rounding arithmetic...

$$p_0^* = 1.0000, \quad p_1^* = 0.33333$$

which modifies

$$c_1^* = 1.0000, \quad c_2^* = -0.12500 \cdot 10^{-5}$$

The generated sequence is

$$p_n^* = 1.0000\,(0.33333)^n - \underbrace{0.12500 \cdot 10^{-5} (3.0000)^n}_{\text{Exponential Growth}}$$

p_n^* quickly becomes a very poor approximation to p_n due to the exponential growth of the initial roundoff error.

The effects of roundoff error can be reduced by using higher-order-digit arithmetic such as the **`double`** or multiple-precision arithmetic available on most computers.

Disadvantages in using **`double`** precision arithmetic are that it takes more computation time and ***the growth of the roundoff error is not eliminated but only postponed.***

Sometimes, but not always, it is possible to reduce the growth of the roundoff error by restructuring the calculations.

Definition: ***Rate of Convergence —***

Suppose the sequence $\underline{\beta} = \{\beta_n\}_{n=1}^{\infty}$ converges to zero, and $\underline{\alpha} = \{\alpha_n\}_{n=1}^{\infty}$ converges to a number α.

If $\exists K > 0$: $|\alpha_n - \alpha| < K\beta_n$, for n large enough, then we say that $\{\alpha_n\}_{n=1}^{\infty}$ converges to α with a **Rate of Convergence** $\mathcal{O}(\beta_n)$ ("Big Oh of β_n").

We write

$$\alpha_n = \alpha + \mathcal{O}(\beta_n)$$

Note: The sequence $\underline{\beta} = \{\beta_n\}_{n=1}^{\infty}$ is usually chosen to be

$$\beta_n = \frac{1}{n^p}$$

for some positive value of p.

Example #1

If

$$\alpha_n = \alpha + \frac{1}{\sqrt{n}}$$

then for any $\epsilon > 0$

$$|\alpha_n - \alpha| = \frac{1}{\sqrt{n}} \le \underbrace{(1+\epsilon)}_{K} \underbrace{\frac{1}{\sqrt{n}}}_{\beta_n}$$

hence

$$\alpha_n = \alpha + \mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$$

Example #2: Consider the sequence (as $n \to \infty$)

$$\alpha_n = \sin\left(\frac{1}{n}\right) - \frac{1}{n}$$

We Taylor expand $\sin(x)$ about $x_0 = 0$:

$$\sin\left(\frac{1}{n}\right) \sim \frac{1}{n} - \frac{1}{6n^3} + \mathcal{O}\left(\frac{1}{n^5}\right)$$

Hence

$$|\alpha_n| = \left|\frac{1}{6n^3} + \mathcal{O}\left(\frac{1}{n^5}\right)\right|$$

It follows that

$$\alpha_n = \mathbf{0} + \mathcal{O}\left(\frac{1}{n^3}\right)$$

Note:

$$\mathcal{O}\left(\frac{1}{n^3}\right) + \mathcal{O}\left(\frac{1}{n^5}\right) = \mathcal{O}\left(\frac{1}{n^3}\right), \quad \text{since} \quad \frac{1}{n^5} \ll \frac{1}{n^3}, \quad \text{as} \quad n \to \infty$$

Definition: ***Rate of Convergence —***

Suppose

$$\lim_{h \searrow 0} G(h) = 0, \quad \text{and} \quad \lim_{h \searrow 0} F(h) = L$$

If $\exists K > 0$:

$$|F(h) - L| \le K\,|G(h)|$$

$\forall h < H$ (for some $H > 0$), then

$$F(h) = L + \mathcal{O}(G(h))$$

we say that $F(h)$ converges to L with a **Rate of Convergence** $\mathcal{O}(G(h))$.

Usually $G(h) = h^p$, $p > 0$.

Example #2-b: Consider the function $\alpha(h)$ (as $h \to 0$)

$$\alpha(h) = \sin(h) - h$$

We Taylor expand $\sin(x)$ about $x_0 = 0$:

$$\sin(h) \sim h - \frac{h^3}{6} + \mathcal{O}\left(h^5\right)$$

Hence

$$|\alpha(h)| = \left| \frac{h^3}{6} + \mathcal{O}\left(h^5\right) \right|$$

It follows that

$$\lim_{h \to 0} \alpha(h) = \mathbf{0} + \mathcal{O}\left(h^3\right)$$

Note:

$$\mathcal{O}\left(h^3\right) + \mathcal{O}\left(h^5\right) = \mathcal{O}\left(h^3\right), \quad \text{since} \quad h^5 \ll h^3, \quad \text{as} \quad h \to 0$$

Our new favorite problem:

$$f(x) = 0$$

We are going to solve the equation $f(x) = 0$ (*i.e.* finding root to the equation), for functions f that are complicated enough that there is no closed form solution (and/or we are too lazy to find it?)

In a lot of cases we will solve problems to which we can find the closed form solutions — we do this as a training ground and to evaluate how good our numerical methods are.

Suppose f is continuous on the interval (a_0, b_0) and $f(a_0) \cdot f(b_0) < 0$ — This means the function changes sign at least once in the interval.

The **intermediate value theorem** guarantees the existence of $c \in (a_0, b_0)$ such that $f(c) = 0$.

Without loss of generality (just consider the function $-f(x)$), we can assume (for now) that $f(a_0) < 0$.

We will construct a sequence of intervals containing the root c:

$$(a_0, b_0) \supset (a_1, b_1) \supset \cdots \supset (a_{n-1}, b_{n-1}) \supset (a_n, b_n) \ni c$$

The sub-intervals are determined recursively:

Given (a_{k-1}, b_{k-1}), let $m_k = \dfrac{a_{k-1} + b_{k-1}}{2}$ be the mid-point.

If $f(m_k) = 0$, we're done, otherwise

$$(a_k, b_k) = \begin{cases} (m_k, b_{k-1}) & \text{if } f(m_k) < 0 \\ (a_{k-1}, m_k) & \text{if } f(m_k) > 0 \end{cases}$$

This construction guarantees that $f(a_k) \cdot f(b_k) < 0$ and $c \in (a_k, b_k)$.

After n steps, the interval (a_n, b_n) has the length

$$|b_n - a_n| = \left(\frac{1}{2}\right)^n |b_0 - a_0|$$

we can take

$$m_{n+1} = \frac{a_n + b_n}{2}$$

as the estimate for the root c and we have

$$c = m_{n+1} \pm d_n, \quad d_n = \left(\frac{1}{2}\right)^{n+1} |b_0 - a_0|$$

Convergence is slow:

At each step we gain ***one binary digit in accuracy.*** Since $10^{-1} \approx 2^{-3.3}$, it takes on average 3.3 iterations to gain one decimal digit of accuracy.

Note: The rate of convergence is completely independent of the function f.

We are only using the **sign of** f at the endpoints of the interval(s) to make decisions on how to update. — By making more effective use of the values of f we can attain significantly faster convergence.

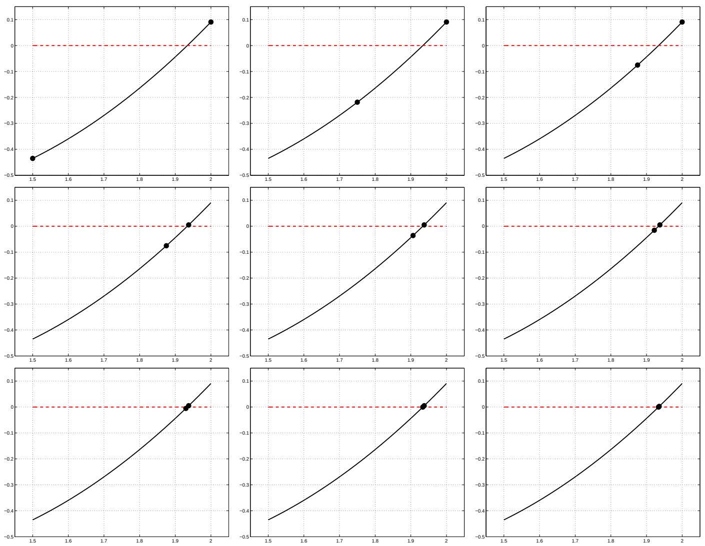
First an example...

The bisection method applied to

$$f(x) = \left(\frac{x}{2}\right)^2 - \sin(x) = 0$$

with $(a_0, b_0) = (1.5, 2.0)$, and $(f(a_0), f(b_0)) = (-0.4350, 0.0907)$ gives:

k	a_k	b_k	m_{k+1}	$f(m_{k+1})$
0	1.5000	2.0000	1.7500	–0.2184
1	1.7500	2.0000	1.8750	–0.0752
2	1.8750	2.0000	1.9375	0.0050
3	1.8750	1.9375	1.9062	–0.0358
4	1.9062	1.9375	1.9219	–0.0156
5	1.9219	1.9375	1.9297	–0.0054
6	1.9297	1.9375	1.9336	–0.0002
7	1.9336	1.9375	1.9355	0.0024
8	1.9336	1.9355	1.9346	0.0011
9	1.9336	1.9346	1.9341	0.0004




```
% WARNING: This example ASSUMES that f(a)<0<f(b)...

x = 1.5:0.001:2;
f = inline('(x/2).^2-sin(x)','x');

a = 1.5;
b = 2.0;

for k = 0:9
   plot(x,f(x),'k-','linewidth',2)
   axis([1.45 2.05 -0.5 .15])
   grid on
   hold on
   plot([a b],f([a b]),'ko','linewidth',5)
   hold off
   m = (a+b)/2;
   if( f(m) < 0 )
      a = m;
   else
      b = m;
   end
   pause
   print('-depsc',['bisec' int2str(k) '.eps'],'-f1');
end
```


Stopping Criteria

When do we stop?

We can *(1)* keep going until successive iterates are close:

$$|m_{k+1} - m_k| < \epsilon$$

or *(2)* close in relative terms

$$\frac{|m_{k+1} - m_k|}{|m_{k+1}|} < \epsilon$$

or *(3)* the function value is small enough

$$|f(m_{k+1})| < \epsilon$$

No choice is perfect. In general, where no additional information about f is known, the second criterion is the preferred one (since it comes the closest to testing the relative error).