Numerical Analysis

A. DONEV

Instead of trying to define what numerical analysis is, let's try to compute square roots.

Take $\begin{cases} c \geq 0 \\ x = \sqrt{c} \end{cases} \Rightarrow x^2 = c$

nonlinear equation (first few lectures)

How does a calculator/computer compute $\sqrt{2}$ (or π)?

Assume the machine can add/subtract and multiply (easy) and divide (not as easy) "real" numbers (quotes explained in a few lectures)

How do we compute $\sqrt{c>0}$?

Wikipedia gives the ancient Babylonian method

(1) $$x_{k+1} = \frac{1}{2}\left(x_k + \frac{c}{x_k}\right), \quad k = 1, 2, \ldots \infty$$

($k+1$: index)

Recursion or iteration so this is an iterative method (almost all methods we will study are iterative)

Choose initial guess $x_0 > 0$

[Matlab demo Babylonian.m]

Theorem: Babylonian method converges to $\sqrt{c}$,

$$\lim_{k\to\infty} x_k = \sqrt{c}$$

(shorthand $x_k \xrightarrow[k\to\infty]{} c$)

for any $x_0 > 0$.

We can actually generalize the method to be

$$X_{k+1} = pX_k + (1-p)\frac{c}{X_k} \quad (2)$$

$$0 < p < 1$$

(Q1) [(think about why $p=0$ and $p=1$ are no good & discuss)]

For $p = 1/2$ we get Babylon. (1)

Iteration (2) is an example of a fixed-point method, which we will study further next class.

To see why this may work, imagine the method converges, i.e. $X_k \to \tilde{X}$

$$\Rightarrow X_{k+1} = p\tilde{X} + (1-p)\frac{c}{\tilde{X}} = \tilde{X}$$

$$\Rightarrow (1-p)\frac{c}{\tilde{X}} = (1-p)\tilde{X} \quad (3)$$

$\Rightarrow \quad \tilde{x}^2 = c \quad \Rightarrow \tilde{x} = \sqrt{c}$

as desired. So if the method converges it will converge to the square root. We say that (2) is a **consistent method**

(method is consistent with equation)

Method (2) is not only consistent but it is also convergent (not yet proven), which means it is OK to use it - it will give the correct answer.

But is it a good method?

What does "good" mean?

How fast does the fixed-point method (2) converge?

We will give a precise answer & theorems soon, for now, let's try it out

[Matlab demo BabylonianPlot.m ($p=1/2$) & NonBabylonian.m ($p=1/4, 3/4$)]

absolute error $\leftarrow$ $x_k - \sqrt{c}$

relative error (in %) $\varepsilon_k = \left| \dfrac{x_k - \sqrt{c}}{\sqrt{c}} \right|$

Answer: Method converges
- "very" fast for $p=1/2$, but
- not so fast for $p \neq 1/2$

Later we will explain this, and learn that in general fixed-point methods are linearly convergent but Babylonian method is an example of Newton's method which is quadratically convergent

But how good does our initial guess have to be for the method to converge?

We say, is this a robust method?

For this, we need to do some (numerical) analysis.

Yet the relative error be

$$\varepsilon_k = \frac{x_k - \sqrt{c}}{\sqrt{c}} = \frac{x_k}{\sqrt{c}} - 1$$

$$\Rightarrow \quad x_k = \sqrt{c}\,(1 + \varepsilon_k)$$

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{c}{x_k}\right) \quad \text{(Babylon)}$$

$$\Rightarrow \varepsilon_{k+1} = \frac{x_{k+1}}{\sqrt{c}} - 1 =$$

$$= \frac{1}{2\sqrt{c}} \left[(1+\varepsilon_k)\sqrt{c} + \frac{c}{(1+\varepsilon_k)\sqrt{c}} \right] - 1$$

$$= \frac{1}{2} \left((1+\varepsilon_k) + \frac{1}{(1+\varepsilon_k)} \right) - 1$$

$$= \frac{(1+\varepsilon_k)^2 + 1 - 2(1+\varepsilon_k)}{2(1+\varepsilon_k)} \Rightarrow$$

$$\boxed{\varepsilon_{k+1} = \frac{\varepsilon_k^2}{2(1+\varepsilon_k)} > 0} \quad \text{for } k \geq 0$$

since $\varepsilon_0 > -1$

Since $\varepsilon_k > 0 \Rightarrow \varepsilon_{k+1} < \frac{\varepsilon_k}{2}$

$$\varepsilon_{k+1} = \frac{\varepsilon_k}{1+\varepsilon_k} \cdot \frac{\varepsilon_k}{2} < \frac{\varepsilon_k}{2}$$

$$\Rightarrow \varepsilon_{k+1} \leq \min \left\{ \frac{\varepsilon_k}{2}, \frac{\varepsilon_k^2}{2} \right\}$$

$$\varepsilon_{k+1} \leq \frac{\varepsilon_k}{2} \quad \text{[convergence]}$$

If $\frac{\varepsilon_k}{2} < \frac{\varepsilon_k^2}{2} \Rightarrow \varepsilon_k > 1$

then $\varepsilon_{k+1} \leq \varepsilon_k/2 < \varepsilon_k$

If $\varepsilon_k < 1 \Rightarrow$

$$\varepsilon_{k+1} \leq \frac{\varepsilon_k^2}{2} < \frac{\varepsilon_k}{2}$$

Summary: When error is large (larger than 100%), it is at least halved in the next step, so it decreases, albeit slowly (linearly). Once error drops below 100%, it gets squared and divided by 2, so decreases faster. Once $\varepsilon_k << 1$, $\varepsilon_k^2/2 << \varepsilon_k$ and error decreases fast (quadratically)

This explains what we saw in Matlab for a bad initial guess (large mitial error)

Is this the complete story of the numerical analysis of the Babylonian method?

Not quite, but the last step is still a bit elusive.

Let's look at another method from Wikipedia instead, the "two variable method"

$$x_0 = c \quad , \quad y_0 = c - 1$$

$$c \in (0, 3)$$

$$\begin{cases} x_{k+1} = x_k - \dfrac{x_k y_k}{2} \\ y_{k+1} = y_k^2 \, (y_k - 3)/4 \end{cases} \qquad (3)$$

(9)

(Q2) Theorem (see Wiki & try to work through calculations)

$$\begin{cases} x_k \to \sqrt{c} \\ y_k \to 0 \end{cases}$$ (quadratically fast)

Method (3) only requires dividing by 2 & 4 which is just moving a decimal place in base 2 so this method was used on first electronic computers (no fast division)

[Matlab demo SqrtTwoVar.m]

Conclusion: Converges slow at first if c is close to 3 (or $\emptyset$, check!) but then fast at some point. The closer c is to 3 the fewer digits of accuracy we get

We seem to get ~16 digits correctly for c not close to 3. Why?

Why do we loose digits for $c = 2.999$ for example?

We will partially understand this once we talk about floating-point arithmetic & round off error & stability (it turns out iteration (3) has unstable error propagation)

The study of algorithms in terms of consistency, stability, and convergence, as well as their (computational) efficiency and robustness, is called "numerical analysis". (11)