Numerical Methods

Compiled by Tristan Pang

11/2023

These notes are based on the Oxford Numerical Methods course taught by David Marshall (2023 for the NERC DTP) with additional information from LeVeque [1].

These are rough notes only. A polished version may or may not be completed. Please direct all typos to me. The GitHub repo[1] contains LaTeX source, Python scripts for figures, and other useful Python things.

Contents

1 Root finding **1**

1.1 Bisection method . . . 2

1.2 Newton's method . . . 3

1.3 Higher dimensions . . . 4

2 Finite difference **5**

3 Von Neumann analysis **5**

4 Numerical linear algebra **5**

A Background theory **5**

A.1 Big O notation . . . 5

A.2 Taylor expansions . . . 5

References **5**

1 Root finding

Consider a sufficiently smooth function $f(x)$. If f is a quadratic polynomial, we may find the zeros of f using the quadratic formula, but for degrees 5 or larger, there exists no general formula for the zeros (Abel–Ruffini theorem). In general, finding an x^* such that $f(x^*) = 0$ cannot be computed exactly. Instead one must employ numerical root finding algorithms. Common methods include the bisection method and Newton's method.

[1] https://github.com/tristanpang/numerical-methods-notes

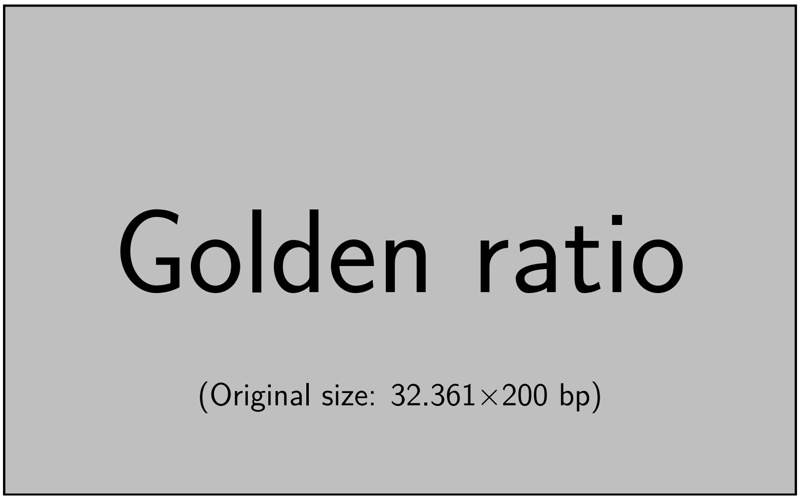

Figure 1: Bisection method

1.1 Bisection method

To find a zero x^* of f, the bisection method takes two initial guesses a and b such that $a < 0$ and $b \geq 0$. IVT guarantees a zero between the two guesses. Calculate the midpoint

$$c = \frac{a+b}{2}.$$

If $f(c) < 0$, replace a with c; otherwise replace b with c. Continue iterating until convergence is observed as shown in Figure 1.

The error at the first iteration is

$$\begin{aligned}
\varepsilon_1 &= |c - x^*| \\
&= \left| \frac{a - x^*}{2} + \frac{b - x^*}{2} \right| \\
&= \left| \left| \frac{a - x^*}{2} \right| - \left| \frac{b - x^*}{2} \right| \right| \\
&\leq \left| \frac{a - x^*}{2} - \frac{b - x^*}{2} \right| \\
&= \frac{|b - a|}{2}.
\end{aligned}$$

Thus, in general

$$\varepsilon_n = |c - x^*| \leq \frac{|b - a|}{2},$$

i.e. the error is at least halved each iteration, and the method converges linearly.

Example 1.1
The positive zero of the polynomial $f(x) = x^2 - 2$ can be approximated using bisection with starting guesses $a = 1$ and $b = 2$. Then $f(1) = -1 < 0$ and $f(2) = 2 > 0$. It follows (by continuity of f) that there is a root in the interval $[1, 2]$. Then $f(c) = f(1.5) = 0.25 > 0$. Thus, we replace $b = 2$ with $c = 1.5$. Continuing yields $1.25, 1.375, 1.4375, \ldots$. Eventually, we get an approximation for $\sqrt{2}$.

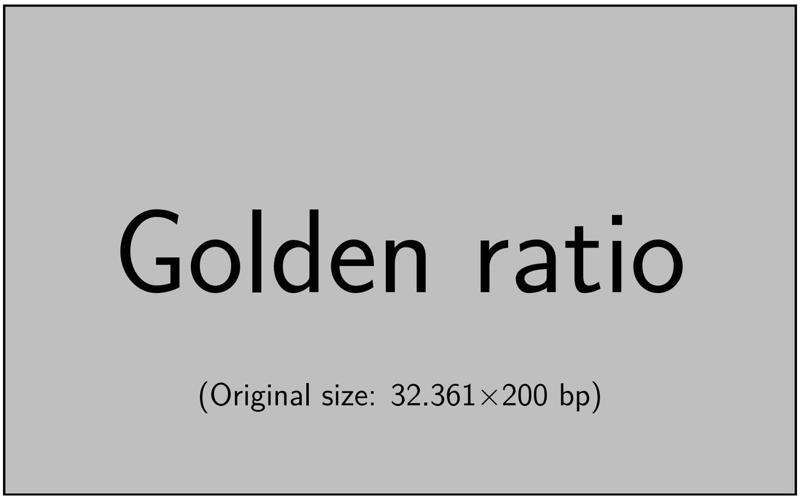

Figure 2: Newton's method

1.2 Newton's method

A quicker alternative to bisection is Newton's method (also known as Newton-Raphson). Given an initial guess x_0 and a sufficiently nice derivative f', we may estimate a zero x^* of f.

Consider the Taylor expansion of f around x_n (see Appendix A.1 for the big O notation and Appendix A.2 for Taylor):

$$f(x) = f(x_n) + f'(x_n)(x - x_n) + O((x - x_n)^2).$$

If we suppose that x_n is close to the root x^*, the zero of the linear approximation x_{n+1} is a good approximation for x^*

$$f(x_{n+1}) \approx 0 = f(x_n) + (x_{n+1} - x_n)f'(x_n).$$

Rearranging, we arrive at the iterative formula for Newton's method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{1}$$

This process is illustrated in Figure 2.

The signed error at iteration n is $\varepsilon_n = x_n - x^*$. By considering the quadratic term in the Taylor expansion around x_n, we get

$$\begin{aligned}
f(x^*) &= f(x_n) + f'(x_n)(x^* - x_n) + \frac{f''(x_n)}{2}(x^* - x_n)^2 + O((x^* - x_n)^3) \\
\implies \quad 0 &= f(x_n) + f'(x_n)(x^* - x_n) + \frac{f''(x_n)}{2}\varepsilon_n^2 + O(\varepsilon_n^3) \\
\implies \quad -\frac{f(x_n)}{f'(x_n)} &= (x^* - x_n) + \frac{f''(x_n)}{2}\varepsilon_n^2 + O(\varepsilon_n^3) \\
\implies \quad x_{n+1} - x_n &= (x^* - x_n) + \frac{f''(x_n)}{2}\varepsilon_n^2 + O(\varepsilon_n^3) \\
\implies \quad \varepsilon_{n+1} &= \frac{f''(x_n)}{2}\varepsilon_n^2 + O(\varepsilon_n^3)
\end{aligned}$$

Thus, as $n \to \infty$, $x_n \to x^*$ for a root x^* of f. In particular, we have quadratic convergence.

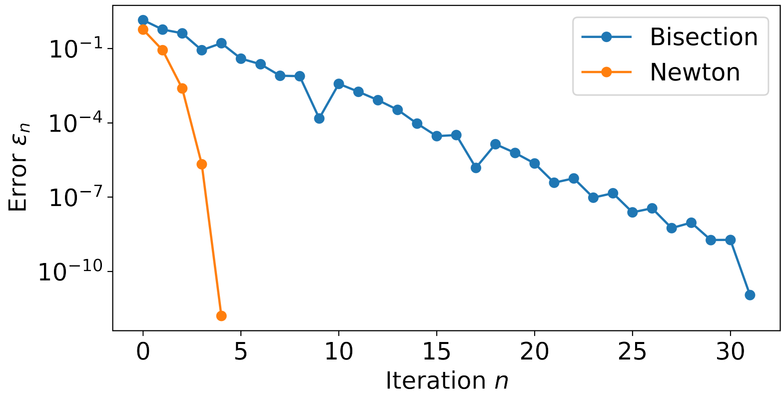

Figure 3: Bisection method vs Newton's method for approximating $\sqrt{2}$

Example 1.2
The positive zero of the polynomial $f(x) = x^2 - 2$ can be approximated using Taylor's method with the starting guess $x_0 = 2$. Differentiating, $f'(x) = 2x$. Then we get

$$\begin{aligned} x_1 &= x_0 - \frac{f(x_0)}{f'(x_0)} = 2 - \frac{2^2 - 2}{4} = 1.5, \\ x_2 &= 1.5 - \frac{1.5^2 - 2}{3} = 1.41666667, \\ x_3 &= 1.41421569, \\ x_4 &= 1.41421356 \approx \sqrt{2}. \end{aligned}$$

This converges to $\sqrt{2}$ much faster than the bisection method as seen in Figure 3.

Exercise 1.3
Observe (in Python or otherwise) that approximating $\sqrt{2}$ with a bisection guess of $(1, 200)$ and Newton guess of 200 yields similar log-linear error behaviour for small iteration step n. Show that this is true by looking at Formula 1.

Warning: if $f'(x_n) = 0$, Newton's method will not work (division by zero!) – pick a new x_0. If the derivative is not well behaved (either not defined or close to zero at many points), then Newton's method may not be appropriate.

1.3 Higher dimensions

Consider the system of m equations in n variables $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ given by

$$\begin{cases} f_1(x_1, x_2, \dots, x_n) = 0, \\ f_2(x_1, x_2, \dots, x_n) = 0, \\ \qquad\qquad \vdots \\ f_m(x_1, x_2, \dots, x_n) = 0. \end{cases}$$

The iterative step of Newton's method becomes

$$\mathbf{x}_{n+1} = \mathbf{x}_n - J(\mathbf{x}_n)^{-1}\mathbf{f}(\mathbf{x}_n), \tag{2}$$

where J is the Jacobian matrix of $\mathbf{f}$ (an analogue to the derivative) given by $J_{ij} = \frac{\partial f_i}{\partial x_j}$. This requires either matrix inversion (which is usually hard!) or solving a linear system (see Section 4).

Example 1.4
Consider the steady state of the predator prey model:

$$\begin{cases} f_1(x,y) = Ax - Bxy = 0, \\ f_2(x,y) = Dxy - Cy = 0. \end{cases}$$

The Jacobian is

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{pmatrix} = \begin{pmatrix} A - By & -Bx \\ Dy & Dx - C \end{pmatrix}.$$

Let $\mathbf{x}_0 = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$. Then

$$\mathbf{x}_1 = \mathbf{x}_0 - J(\mathbf{x}_0)^{-1}\mathbf{f}(\mathbf{x}_n) = \begin{pmatrix} 2 \\ 1 \end{pmatrix} - \begin{pmatrix} A - B & -2B \\ D & 2D - C \end{pmatrix}^{-1} \begin{pmatrix} 2A - 2B \\ 2D - C \end{pmatrix}.$$

Note 1.5 (Useful commands)

- Python SciPy's `otimize.fsolve` finds the roots of a function.
- Python NumPy's `linalg.solve` solves a linear system.

2 Finite difference

3 Von Neumann analysis

4 Numerical linear algebra

A Background theory

A.1 Big O notation

A.2 Taylor expansions

Theorem A.1 (Taylor)

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \cdots$$

References

[1] R. J. LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 2007.