Metodos Numericos en Astronomia Teorica, lecture 2

Dominik Schleicher

Universidad de Concepcion

April 12, 2020

Good programming style (1)

- Write Clearly - don't be too clever - don't sacrifice clarity for efficiency.
- Say what you mean, simply and directly.
- Be sparing with temporary variables.
- Parenthesize to avoid ambiguity.
- Use library functions.
- Replace repetitive expressions by calls to a common function.
- Choose variable names that won't be confused.
- If a logical expression is hard to understand, try transforming it.
- Choose a data representation which makes the program simple.
- Don't patch bad code - rewrite it.
- Write and test a big program in small pieces.

- Test input for plausibility and validity.
- Identify bad input - recover if possible.
- Make sure input doesn't violate the limits of the program.
- Terminate input by end-of-file or marker, not by count.
- Make input easy to prepare and output self-explanatory.

See http://www.eg.bucknell.edu/~xmeng/Course/CS2330/Handout/StyleKP.html

Good programming style (2)

- Make sure all variables are initialized before use.
- Watch out for off-by-one errors.
- Make sure your program "does nothing" gracefully.
- Test programs at their boundary values.
- Check some answers by hand.
- 10.0 times 0.1 is hardly ever 1.0.
- Don't compare floating point numbers solely for equality.

- Make it right before you make it faster.
- Make it fail-safe before you make it faster.
- Make it clear before you make it faster.
- To make it faster, change the algorithm not small details in the code.
- Actually test code to see how fast it is.

See http://www.eg.bucknell.edu/~xmeng/Course/CS2330/Handout/StyleKP.html

Good programming style (3)

- Make sure comments and code agree.
- Use variable names that mean something.
- Format a program to help the reader understand it.
- Don't just echo code in comments - make every comment meaningful.
- Document your data structures.
- Don't over comment.
- Don't comment bad code - rewite it.

- Use recursive procedures for recursively defined data structures.
- Use data arrays to avoid repetitive control sequences.

See http://www.eg.bucknell.edu/~xmeng/Course/CS2330/Handout/StyleKP.html

Error analysis

- An abstract problem setup consists of some data x and a desired result $f(x)$, defined as a method $f : x \longmapsto f(x)$.
- In a real problem, we have data $\boxed{x}$ with limited accuracy. In addition, the method f cannot be applied with infinite accuracy, due to
  - Limited accuracy $\Rightarrow$ round-off errors,
  - limited space $\Rightarrow$ approximation errors,
  - limited time $\Rightarrow$ methodological errors.
- In general, one thus employs an approximate method $\boxed{f}$, leading to an approximate result $\boxed{f}(\boxed{x})$.
- The approximation and methodological errors need to be discussed for each method separately.

Representation of real numbers

- The real numbers are an uncountably infinite set - every representation must be approximate.

- Today it is common to adopt a floating point representation, given as

$$x = a \times 2^e, \tag{1}$$

with

$$e \in \{e_{min}, ..., e_{max}\} \tag{2}$$

$$a = v \sum_{i=1}^{l} a_i b^{-i} \quad \text{or } 0. \tag{3}$$

- Here, $v = \pm 1$ denotes the sign, and $a_i = 0, 1$.

- It is convention that $a_1 = 1$ (uniqueness of representation).

Advantages of Floating Point Representation

- The relative accuracy of floating point representation is independent of x.
- All numbers between a minimum of $|x| = 2^{e_{min}-1}$ and a maximum of $|x| = 2^{e_{max}}(1-2^{-l})$ can be represented with an accuracy of

$$\frac{|x - \boxed{x}|}{|x|} < \frac{\epsilon}{2}, \tag{4}$$

with $\epsilon = 2^{1-l}$.

- Numbers with $|x| > 2^{e_{max}}(1-2^{-l})$ are said to cause overflow, numbers with $|x| < 2^{e_{min}-1}$ are said to cause underflow.

Realizations of Floating Point Representation

- Modern programming languages distinguish data types with single precision and double precision:

Level	Width	Range at full precision	Precision[a]
Single precision	32 bits	$\pm 1.18 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$	Approximately 7 decimal digits
Double precision	64 bits	$\pm 2.23 \times 10^{-308}$ to $\pm 1.80 \times 10^{308}$	Approximately 16 decimal digits

Error propagation

- The relative representation error $\epsilon/2$ will propagate due to numerical operations.
- For standard operations $\oslash \in \{+, -, \cdot, /\}$, we have

$$a \boxed{\oslash} b = (a \oslash b)(1 + \tilde{\epsilon}), \quad \text{with } \tilde{\epsilon} < \epsilon. \tag{5}$$

- In general, error propagation can be described as a random walk. After N operations, the relative accuracy is thus $\sqrt{N}\epsilon$.

Numerical integration

- The problem referred to as numerical integration is to provide an approximate solution to a definite integral of the form

$$I(a, b) = \int_a^b f(x) dx. \tag{6}$$

- The mathematical definition of an integral is the limit of the sum over boxes as their width h approaches zero:

$$\int_a^b f(x) dx = \lim_{h \to 0} \left[h \sum_{i=1}^{(b-a)/h} f(x_i) \right] \tag{7}$$

Numerical integration

- Numerically, the integral is approximated as a finite sum over boxes:

$$\int_a^b f(x)dx \sim \sum_{i=1}^{N} f(x_i)\omega_i. \tag{8}$$

The function f is thus discretized into values $f_i = f(x_i)$ at the points x_i, with ω_i denoting an appropriate weight at point x_i.

- One possibile choice for the discretization is to adopt

$$\omega_i = h := \frac{b-a}{N}, \quad x_{i+1} = x_i + h. \tag{9}$$

- This approach is however crude and requires a small spacings h. We will thus consider more accurate approaches in the following.

Numerical integration

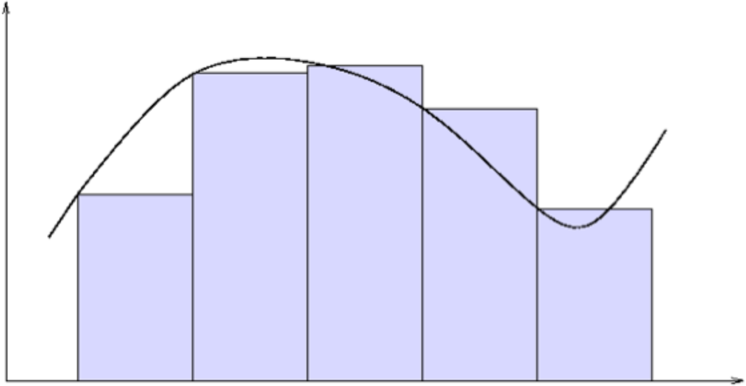

Fig. 1: Approximation via rectangles.

Numerical integration

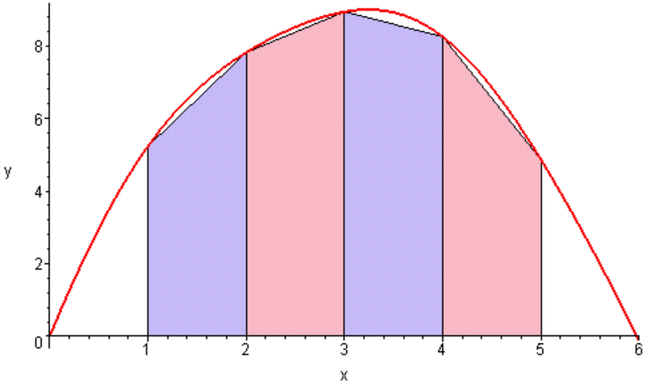

Fig. 2: First-order improvement: Trapezoid rule.

Trapezoid rule

- Trapezoid rule: N evenly spaced points x_i, $N-1$ length intervals h:

$$h = \frac{b-a}{N-1} \quad (10)$$
$$x_i = a + (i-1)h, \quad i = 1, N \quad (11)$$

- We construct a trapezoid of width h in each interval i, consisting of $(x_i, 0)$, $(x_{i+1}, 0)$, $(x_{i+1}, f(x_{i+1}))$, $(x_i, f(x_i))$.

- The area of a single trapezoid is then

$$\int_{x_i}^{x_{i+1}} f(x)dx \sim \frac{h(f_i + f_{i+1})}{2} = \frac{1}{2}hf_i + \frac{1}{2}hf_{i+1} \quad (12)$$

- For $N = 2$ points, the weights are thus $\omega_1 = \omega_2 = \frac{1}{2}h$.

Trapezoid rule

- Summing up the trapezoids over the entire interval $[a, b]$ yields

$$\int_a^b f(x)dx \sim \frac{h}{2}f_1 + hf_2 + hf_3 + ... + hf_{N-1} + \frac{h}{2}f_N. \qquad (13)$$

- The internal points are counted twice and thus obtain weight h, while the endpoints have weight $h/2$. Thus,

$$\omega_i = \left\{\frac{h}{2}, h, ..., h, \frac{h}{2}\right\}. \qquad (14)$$

Simpson's rule

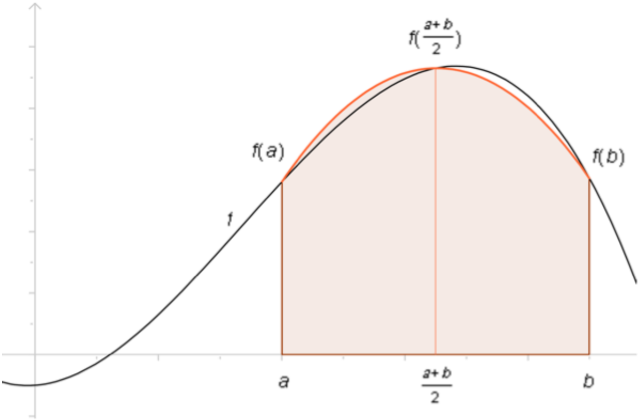

Fig. 3: Second-order improvement: Simpson's rule.

Simpson's rule

- As for the trapezoidal rule, one adopts N points with equal spacing h. Here, N needs to be an odd number.
- In each interval, the function f is now approximated by a parabola

$$f(x) \sim \alpha x^2 + \beta x + \gamma. \tag{15}$$

- The area of each section is now the integral of such a parabola,

$$\int_{x_i}^{x_{i+1}} (\alpha x^2 + \beta x + \gamma) dx = \frac{\alpha x^3}{3} + \frac{\beta x^2}{2} + \gamma x \Big|_{x_i}^{x_{i+1}}. \tag{16}$$

- Considering an interval $[-1, 1]$, we have

$$\int_{-1}^{+1} (\alpha x^2 + \beta x + \gamma) dx = \frac{2\alpha}{3} + 2\gamma. \tag{17}$$

Simpson's rule

- Due to the identities

$$
\begin{aligned}
f(-1) &= \alpha - \beta + \gamma, && (18)\\
f(0) &= \gamma, && (19)\\
f(1) &= \alpha + \beta + \gamma, && (20)
\end{aligned}
$$

we have

$$
\begin{aligned}
\alpha &= \frac{f(1) + f(-1)}{2} - f(0), && (21)\\
\beta &= \frac{f(1) + f(-1)}{2}, && (22)\\
\gamma &= f(0). && (23)
\end{aligned}
$$

Simpson's rule

- The integral can thus be expressed as the weighted sum over the function at 3 points:

$$\int_{-1}^{+1} (\alpha x^2 + \beta x + \gamma) dx = \frac{f(-1)}{3} + \frac{4f(0)}{3} + \frac{f(1)}{3}. \tag{24}$$

- As 3 values of the function are needed, the result is generalized by evaluating f over two adjacent intervals:

$$\int_{x_i-h}^{x_i+h} f(x)dx = \int_{x_i}^{x_i+h} f(x)dx + \int_{x_i-h}^{x_i} f(x)dx \sim \frac{h}{3} f_{i-1} + \frac{4h}{3} f_i + \frac{h}{3} f_{i+1}. \tag{25}$$

- As we integrate over pairs of intervals, N needs to be an odd number.

Simpson's rule

- Integrating over the entire interval yields

$$\int_a^b f(x)dx \sim \frac{h}{3}f_1 + \frac{4h}{3}f_2 + \frac{2h}{3}f_3 + \frac{4h}{3}f_4 + ... + \frac{4h}{3}f_{N-1} + \frac{h}{3}f_N. \quad (26)$$

- The integration weights are thus given as

$$\omega_i = \left\{ \frac{h}{3}, \frac{4h}{3}, \frac{2h}{3}, \frac{4h}{3}, ..., \frac{4h}{3}, \frac{h}{3} \right\}. \quad (27)$$

- The sum of the weights can be used to check the integration:

$$\sum_{i=1}^{N} \omega_i = (N-1)h. \quad (28)$$

Integration error

- In general, one aims for a method yielding an accurate answer using the least number of integration points.
- To estimate both the absolute method error E_m and the relative method error ϵ_m, we expand $f(x)$ in a Taylor series around the midpoint of each interval i.
- The total error is then estimated by multiplying with the number of grid points N.
- The trapezoid rule uses linear interpolation for the function f. The inaccuracy in f is thus of order $h^2 f''$, and after integration $h^3 f''$.
- The error in the overall interval is thus of order $N h^3 f'' \sim \frac{(b-a)^3}{N^2} f''$.

Integration error

- For Simpson's rule, f is approximated with a parabolic function.
- The error in approximating f is thus of order $h^3 f^{(3)}$.
- One could thus naively expect an error of $h^4 f^{(3)}$ after integration. However, the third-order terms cancel out, and the dominant error is of order $h^5 f^{(4)}$.
- Multiplying with N, the total integral has an error of order $Nh^5 f^{(4)} \sim \frac{(b-a)^5}{N^4} f^{(4)}$.
- For small intervals h and well-behaved functions f, the Simpson's rule should converge more rapidly than the trapezoid rule.

Integration error

- The error discussed above is due to the approximation of f, yielding a relative method error is given as $\epsilon_m = E/f$.
- In addition, there is an accumulating round-off error due to the finite machine precision ϵ_{mp}.
- We assume after N steps, the relative round-off error is random and of the form

$$\epsilon_{ro} = \sqrt{N}\epsilon_{mp} \tag{29}$$

  (random walk).
- For single precision, we have $\epsilon_{mp} \sim 10^{-7}$ and $\epsilon_{mp} \sim 10^{-15}$ for double-precision.

Integration error

- In the following, we want to determine the N which minimizes the total relative error
$$\epsilon_{tot} = \epsilon_{ro} + \epsilon_m. \tag{30}$$

- As ϵ_{ro} decreases with N, ϵ_{tot} is minimal if both errors become approximately equal:
$$\epsilon_{ro} \sim \epsilon_m \sim \frac{E_m}{f}. \tag{31}$$

- We temporarily adopt

$$b - a = 1 \quad \rightarrow \quad h = \frac{1}{N} \tag{32}$$
$$f^{(n)} \sim \frac{f}{(b-a)^n} \sim f. \tag{33}$$

Integration error - trapezoid rule

- When applied to the trapezoid rule, Eq. (31) yields

$$\sqrt{N}\epsilon_{mp} \sim \frac{f''(b-a)^3}{fN^2} = \frac{1}{N^2}, \tag{34}$$

$$\Rightarrow N \sim \frac{1}{(\epsilon_{mp})^{2/5}}. \tag{35}$$

- For single precision, the optimum number N is thus $N = \frac{1}{h} = (1/10^{-7})^{2/5} = 631$.

- For double precision, the optimum number N equals $N = \frac{1}{h} = (1/10^{-15})^{2/5} = 10^6$.

Integration error - Simpson's rule

- When applied to Simpson's rule, Eq. (31) yields

$$\sqrt{N}\epsilon_{mp} \sim \frac{f^{(4)}(b-a)^5}{fN^4} = \frac{1}{N^4}, \tag{36}$$

$$\Rightarrow N \sim \frac{1}{(\epsilon_{mp})^{2/9}}. \tag{37}$$

- For single precision, the optimum number N is thus $N = \frac{1}{h} = (1/10^{-7})^{2/9} = 36.$

- For double precision, the optimum number N equals $N = \frac{1}{h} = (1/10^{-15})^{2/9} = 2154.$

Integration error - conclusion

- Simpson's rule considerably improves about trapezoid rule, as a high precision is reached for a much smaller amount of steps.
- In fact, Simpson's rule allows to obtain errors rather close to machine precision.
- The best numerical approximation to an integral is not obtained for $N \to \infty$, but for $N \lesssim 1000$.

Newton-Cotes Formulae

- Both the trapezoid rule and Simpson's rule are part of the Newton-Cotes formulae, a family of numerical integration techniques.
- General procedure: The interval $[a, b]$ is divided into n equal parts of width $h = (b - a)/n$, with the definitions

$$x_{n+1} = x_n + h, \tag{38}$$
$$f_n = f(x_n). \tag{39}$$

- The function f is approximated by a Lagrange interpolating polynomial.
- Newton-Cotes formulae are called "closed" if the end points $[x_1, x_n]$ are considered, and "open" otherwise.
- While many different Newton-Cotes formulae exist, the Simpson rule is usually sufficient for practical purposes.

Newton-Cotes Formulae

The 4-point closed rule is Simpson's 3/8 rule,

$$\int_{x_1}^{x_4} f(x)\,dx = \frac{3}{8}\,h\,(f_1 + 3\,f_2 + 3\,f_3 + f_4) - \frac{3}{80}\,h^5\,f^{(4)}(\xi)$$

(Ueberhuber 1997, p. 100). The 5-point closed rule is Boole's rule,

$$\int_{x_1}^{x_5} f(x)\,dx = \frac{2}{45}\,h\,(7\,f_1 + 32\,f_2 + 12\,f_3 + 32\,f_4 + 7\,f_5) - \frac{8}{945}\,h^7\,f^{(6)}(\xi)$$

(Abramowitz and Stegun 1972, p. 886). Higher order rules include the 6-point

$$\int_{x_1}^{x_6} f(x)\,dx = \frac{5}{288}\,h\,(19\,f_1 + 75\,f_2 + 50\,f_3 + 50\,f_4 + 75\,f_5 + 19\,f_6) - \frac{275}{12096}\,h^7\,f^{(6)}(\xi),$$

7-point

$$\int_{x_1}^{x_7} f(x)\,dx = \frac{1}{140}\,h(41\,f_1 + 216\,f_2 + 27\,f_3 + 272\,f_4$$

$$+ 27\,f_5 + 216\,f_6 + 41\,f_7) - \frac{9}{1400}\,h^9\,f^{(8)}(\xi),$$

Fig. 4: Newton-Cotes Formulae (source: http://mathworld.wolfram.com).

Root finding

- Root finding: Given a function $f(x)$, we seek x with

$$f(x) = 0. \tag{40}$$

- A general algebraic equation

$$g(x) = h(x) \tag{41}$$

can be solved by looking for roots of the function

$$f(x) = g(x) - h(x). \tag{42}$$

The bisection method

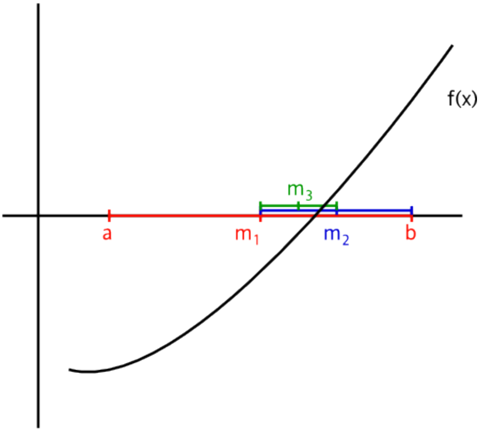

Fig. 5: Finding the root using intervals.

Bisection: Formal approach

- Starting interval $[a_0, b_0]$ with $f(a_0)f(b_0) < 0$ (opposite signs). In step i, we will have some interval $[a_i, b_i]$ with $f(a_i)f(b_i) < 0$.
- Calculate midpoint
$$m_i = \frac{a_i + b_i}{2}. \tag{43}$$
- If $f(m_i)f(a_i) < 0$, set $a_{i+1} = a_i$ and $b_{i+1} = m_i$, otherwise set $a_{i+1} = m_i$ and $b_{i+1} = b_i$.
- Stop iteration once the desired relative error ϵ is reached, i.e.
$$\left| \frac{b_i - a_i}{a_i} \right| < \epsilon. \tag{44}$$

Bisection: Applicability

Bisection is applicable if the function $f(x)$

- is continuous (no jumps).
- has only one root in the interval $[a_0, b_0]$. In case of several roots, the condition $f(a_i)f(b_i) < 0$ may become invalid at some step i and the algorithm breaks down.

Disadvantages:

- requires an appropriate initial guess.
- converges relatively slowly; accuracy improves by a factor of 2 at every step.

Convergence speed (1)

- Assume a sequence x_i. We say that it converges linearly to L if there is a number $\mu \in (0, 1)$ with

$$\lim_{k\to\infty} \frac{|x_{k+1} - L|}{|x_k - L|} = \mu \tag{45}$$

- For $\mu = 0$, the convergence is called superlinear, and for $\mu = 1$, it is sublinear.
- If $\mu = 1$ and

$$\lim_{k\to\infty} \frac{|x_{k+2} - x_{k+1}|}{|x_{k+1} - x_k|} = 1, \tag{46}$$

it converges logarithmically.
- In case of bisection, the accuracy improves by a factor of 2 at every step (linear convergence).

Convergence speed (2)

- If convergence is superlinear, one says that the sequence converges with order $q > 1$ if there is $\mu \in (0, 1)$ with

$$\lim_{k\to\infty} \frac{|x_{k+1} - L|}{|x_k - L|^q} = \mu. \tag{47}$$

- For $q = 2$, we have quadratic convergence, $q = 3$ is called cubic convergence.
- For a general q, we speak of q-linear convergence.
- We now seek methods with faster convergence.

Newton's method

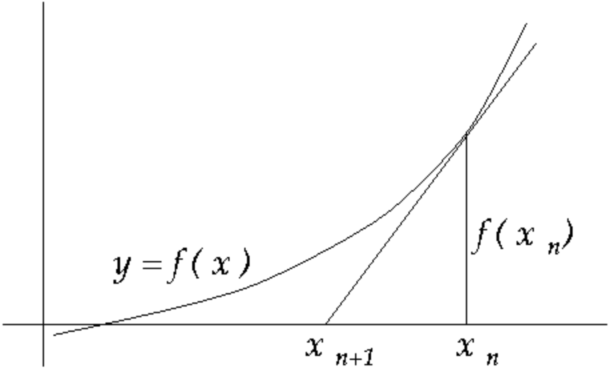

Fig. 6: Idea: Search for root along slope $f'(x)$.

Newton's method

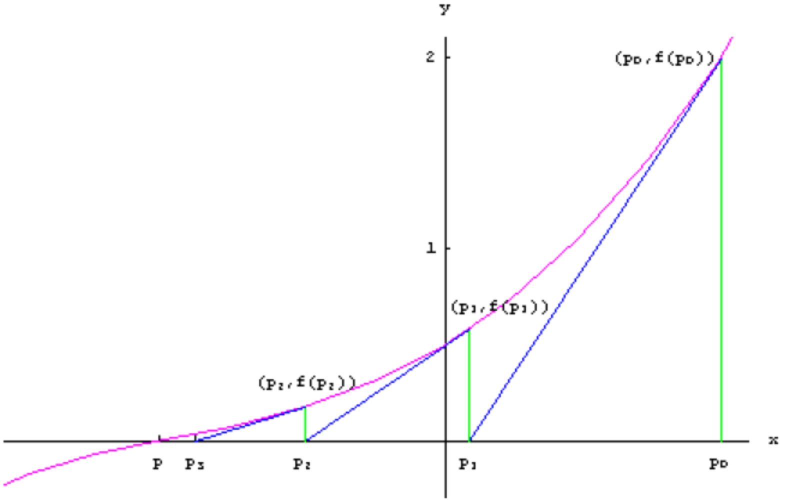

Fig. 7: Iterative convergence.

Newton's method

- Newton's method or Newton-Raphson method:
  Taylor expansion:

$$f(x_0 + \epsilon) = f(x_0) + f'(x_0)\epsilon + \frac{1}{2}f''(x_0)\epsilon^2 + \ldots \tag{48}$$

- To first order, we have

$$f(x_0 + \epsilon) \sim f(x_0) + f'(x_0)\epsilon. \tag{49}$$

- Eq. (49) describes a tangent line to f at $(x_0, f(x_0))$.

- The tangent intersects with the x-axis at

$$\epsilon_0 = -\frac{f(x_0)}{f'(x_0)}, \tag{50}$$

  yielding a first-order guess for the position of the root, $x_1 = x_0 + \epsilon_0$.

Newton's method

- The process is iterated using

$$\epsilon_n = -\frac{f(x_n)}{f'(x_n)}. \tag{51}$$

  The estimated position of the root is then

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{52}$$

- Iteration stops once the desired accuracy is reached.
- The procedure can be unstable near a horizontal asymptote or a local extremum, but otherwise converges for an appropriate initial guess ("approximate zero").

Newton's method

- Assume we have a Newton series x_k converging towards x_* with $f'(x_*) \neq 0$.
- We define the error at step k via

$$x_k = x_* + e_k. \tag{53}$$

- Expanding $f(x_k)$ around x_* yields

$$\begin{aligned} f(x_k) &= f(x_*) + f'(x_*)e_k + \frac{1}{2}f''(x_*)e_k^2 + \dots && (54) \\ &= f'(x_*)e_k + \frac{1}{2}f''(x_*)e_k^2 + \dots && (55) \end{aligned}$$

- Expansion of $f'(x_k)$ yields

$$f'(x_k) = f'(x_*) + f''(x_*)e_k + \dots \tag{56}$$

Newton's method

- We further have

$$e_{k+1} = x_{k+1} - x_* = e_k + (x_{k+1} - x_k) \quad (57)$$

$$= e_k - \frac{f(x_k)}{f'(x_k)} \quad (58)$$

$$\sim e_k - \frac{f'(x_*)e_k + \frac{1}{2}f''(x_*)e_k^2}{f'(x_*) + f''(x_*)e_k}. \quad (59)$$

- For $a \gg b$ and $e \ll 1$, we have the approximate idendity

$$\frac{ae + \frac{1}{2}be^2}{a + be} = \frac{e + \frac{1}{2}be^2/a}{1 + be/a} \quad (60)$$

$$\sim (e + \frac{1}{2}be^2/a)(1 - be/a) \quad (61)$$

$$\sim e - \frac{1}{2}be^2/a - \frac{1}{2}b^2e^2/a^2 \sim e - \frac{1}{2}be^2/a. \quad (62)$$

Newton's method

- Application to Eq. (59) yields the series

$$\epsilon_{k+1} \sim \frac{f''(x_*)}{2f'(x_*)}\epsilon_k^2. \tag{63}$$

- If Newton's method converges, it thus converges quadratically!
- Newton's method is thus the preferred method for root finding; if it does not converge, one may however refer to bisection.

Newton's method 2D

- We extend Newton's method to 2D, from which a further generalization is straightforward.
- This approach can also be used for complex functions, which can always be considered as functions of two arguments x and y.
- We consider the following system of equations:

$$f_1(x,y) = 0, \tag{64}$$
$$f_2(x,y) = 0. \tag{65}$$

- For this system of equations, we define the Jacobian matrix as

$$J(x,y) = \begin{pmatrix} \frac{\partial f_1}{\partial x}(x,y) & \frac{\partial f_1}{\partial y}(x,y) \\ \frac{\partial f_2}{\partial x}(x,y) & \frac{\partial f_2}{\partial y}(x,y) \end{pmatrix}. \tag{66}$$

Newton's method 2D

- We further introduce generalized differentials. For this purpose, we consider the functions

$$u = f_1(x, y), \tag{67}$$
$$v = f_2(x, y). \tag{68}$$

- Their differentials are given as

$$du = \frac{\partial f_1}{\partial x}(x, y)dx + \frac{\partial f_1}{\partial y}(x, y)dy, \tag{69}$$
$$dv = \frac{\partial f_2}{\partial x}(x, y)dx + \frac{\partial f_2}{\partial y}(x, y)dy. \tag{70}$$

- This can be recast as

$$\begin{pmatrix} du \\ dv \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1}{\partial x}(x, y) & \frac{\partial f_1}{\partial y}(x, y) \\ \frac{\partial f_2}{\partial x}(x, y) & \frac{\partial f_2}{\partial y}(x, y) \end{pmatrix} \begin{pmatrix} dx \\ dy \end{pmatrix}. \tag{71}$$

Newton's method 2D

- Introducing

$$d\vec{U} = \begin{pmatrix} du \\ dv \end{pmatrix}, \tag{72}$$

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial x}(x,y) & \frac{\partial f_1}{\partial y}(x,y) \\ \frac{\partial f_2}{\partial x}(x,y) & \frac{\partial f_2}{\partial y}(x,y) \end{pmatrix}, \tag{73}$$

$$d\vec{X} = \begin{pmatrix} dx \\ dy \end{pmatrix}, \tag{74}$$

the generalized differential can be cast as

$$d\vec{U} = J\, d\vec{X}. \tag{75}$$

Newton's method 2D

- We consider the following equation

$$\vec{F}(\vec{X}) = 0, \tag{76}$$

which we aim to solve with Newton's method in 2D. We start with an initial guess $\vec{P}_0 = (p_0, q_0)$. In each step i, we perform the following procedure:

1. Evaluate $\vec{F}(\vec{P}_k)$.
2. Evaluate $J(\vec{P}_k)$.
3. Solve the linear system of equations

$$J(\vec{P}_k)\Delta\vec{P} = -\vec{F}(\vec{P}) \tag{77}$$

   for $\Delta\vec{P}$. In a case of higher dimensionality, this can be done using the Gauss elimination method or more advanced procedures.
4. Continue the iteration with $\vec{P}_{k+1} = \vec{P}_k + \Delta\vec{P}$.

Newton's method 2D

- The method has similar properties as in 1D. In particular, it has a quadratic convergence, and requires that the function has a non-zero derivative at the root.
- However, root finding in 2D and higher dimensions is generally more difficult, especially if several roots are involved. In such cases, already a good initial guess is required, or one must screen through a large parameter space to determine the roots.

The secant method

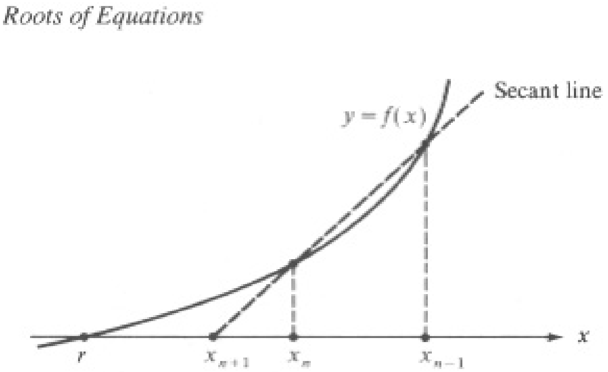

Figure: The secant method for root finding.

The secant method

- The secant method is a straightforward extension of Newton's method, which requires two data points as an initial guess.
- In this method, the derivative of the function is approximated with a difference quotient.
- The iteration procedure is thus

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}. \tag{78}$$

- Convergence is superlinear, but not quadratic. The order of convergences corresponds to the **golden ratio**

$$q = \frac{1+\sqrt{5}}{2} \sim 1.618. \tag{79}$$

- A generalization to higher dimensions is straightforward.
- The secant method predates Newton's method by about 3000 years.