# ***Numerical Analysis***

**Shin, Byeong-Chun**

*Chonnam National University, Gwangju, Korea*

Reference : Elementary Numerical Analysis, Atkinson and Han

# Contents

# 1 Taylor Polynomials

## 1.1 The Taylor Polynomial

– Most function $f(x)$ (e.g. $\cos x, e^x, \sqrt{x}$) cannot be evaluated exactly in simple way.

– The most common classes of approximating function $\hat{f}(x)$ are the polynomials and an efficient approximating polynomial is a Taylor polynomial.

– A related form of function is the piecewise polynomial function.

**Taylor Polynomial** of degree $n$ for a function $f(x)$ about $x = a$ :

- linear polynomial, $p_1(x)$ : $p_1(a) = f(a)$ and $p_1'(a) = f'(a)$

$$p_1(x) = f(a) + (x-a)f'(a).$$

- quadratic polynomial, $p_2(x)$ : $p_2(a) = f(a)$, $p_2'(a) = f'(a)$ and $p_2''(a) = f''(a)$

$$p_2(x) = f(a) + (x-a)f'(a) + \frac{1}{2}(x-a)^2 f''(a).$$

- polynomial of degree $n$, $p_n(x)$ : $p_n(a) = f(a), p_n'(a) = f'(a), \cdots, p_n^{(n)}(a) = f^{(n)}(a)$

$$\left( \text{ i.e., } p_n^{(k)}(a) = f^{(k)}(a), \quad k = 0, 1, \cdots, n \right)$$

$$p_n(x) = f(a) + (x-a)f'(a) + \frac{1}{2}(x-a)^2 f''(a) + \cdots + \frac{(x-a)^n}{n!} f^{(n)}(a) = \sum_{k=0}^{n} \frac{(x-a)^k}{k!} f^{(k)}(a).$$

**Example (1).** Find the Taylor polynomial of degree $n$, $p_n(x; a)$, for $f(x) = e^x$ about $x = a$.

Matlab Code : To compare the three graphs $f(x)$, $p_1(x)$ and $p_2(x)$ around $x = 0$

```
z = -1:0.05:1;                      % z = (x-a) = x
fx = exp(z);                        % function value of f(x)
Dkfzero = ones(1,length(z));        % function value of f^(k)(x) at x=0
p1 = Dkfzero + Dkfzero.*z;          % linear polynomial
p2 = p1 + (1/2)*Dkfzero.*(z.^2);    % quadratic polynomial
plot(z,fx,z,p1,z,p2,':')
```

## 1.2 The Error in Taylor's Polynomial

**Theorem 1.1 (Taylor's Remainder).** *Assume that $f(x)$ has $n+1$ continuous derivatives on an interval $[\alpha, \beta]$, and let $a \in [\alpha, \beta]$. For the Taylor polynomial $p_n(x)$ of $f(x)$, let $R_n(x) := f(x) - p_n(x)$ denote the remainder in approximating $f(x)$ by $p_n(x)$. Then*

$$R_n(x) = \frac{(x-a)^{n+1}}{(n+1)!} f^{(n+1)}(c_x), \quad \alpha \le x \le \beta$$

*with $c_x$ an unknown point between $a$ and $x$.*

**Example (2).** The approximation error of $f(x) = e^x$ and its Taylor polynomial $p_n(x)$ with $a = 0$ is given by

$$e^x - p_n(x) = R_n(x) = \frac{x^{n+1}}{(n+1)!} e^c \quad (n \ge 0) \quad \text{with } c \text{ between } 0 \text{ and } x.$$

For each fixed $x$,

$$e^x - p_n(x) = R_n(x) = \frac{x^{n+1}}{(n+1)!} e^c \longrightarrow 0 \quad \text{as} \quad n \longrightarrow \infty \quad \text{because} \quad \lim_{n\to\infty} \frac{|x|^n}{n!} = 0.$$

Let us take the degree $n$ so that $p_n(x)$ approximates $f(x)$ on an interval $[-1, 1]$ with the accuracy

$$|R_n(x)| \le 10^{-9}.$$

Using the upper bound of $|R_n(x)|$

$$|R_n(x)| = \frac{|x|^{n+1}}{(n+1)!} e^c \le \frac{e}{(n+1)!} < \frac{3}{(n+1)!} \le 10^{-9},$$

we can find the sufficient degree $n$ so that the approximation error is bounded by the tolerance $10^{-9}$ :

$$|R_n(x)| \le 10^{-9} \quad \text{when} \quad n \ge 12.$$

Some Taylor polynomials :

$$e^x = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} + \frac{x^{n+1}}{(n+1)!} e^c,$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots + (-1)^{n-1} \frac{x^{2n-1}}{(2n-1)!} + (-1)^n \frac{x^{2n+1}}{(2n+1)!} \cos c,$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots + (-1)^n \frac{x^{2n}}{(2n)!} + (-1)^{n+1} \frac{x^{2n+2}}{(2n+2)!} \sin c,$$

$$\frac{1}{1-x} = 1 + x + x^2 + \cdots + x^n + \frac{x^{n+1}}{1-x}, \quad (x \ne 1),$$

$$(1+x)^\alpha = 1 + \binom{\alpha}{1} x + \binom{\alpha}{2} x^2 + \cdots + \binom{\alpha}{n} x^n + \binom{\alpha}{n+1} x^{n+1} (1+c)^{\alpha-n-1},$$

where the binomial coefficients are defined by

$$\binom{\alpha}{k} = \frac{\alpha(\alpha-1)\cdots(\alpha-k+1)}{k!}, \quad k = 1, 2, 3, \cdots.$$

Assume that $f(x)$ is infinitely many differentiable at $x = a$ and

$$\lim_{n\to\infty} [f(x) - p_n(x)] = \lim_{n\to\infty} R_n(x) = 0,$$

the infinite series

$$\sum_{k=0}^{\infty} \frac{(x-a)^k}{k!} f^{(k)}(a)$$

is called the **Taylor series expansion** of the function $f(x)$ about $x = a$.

# 2 Error and Computer Arithmetic

## 2.1 Floating-Point Numbers

– Numbers must be stored in computers and arithmetic operations must be performed on these numbers.

– Most computers have two ways of storing numbers, in integer format and in floating-point format.

**Floating-Point Representation in the Decimal system** : more intuitive

$$x = \sigma \cdot \bar{x} \cdot 10^e$$

where $\sigma = +1$ or $-1$ (sign), $e$ is an integer (exponent), and $1 \leq \bar{x} < 10$ (significant or mantissa).

For an example,

$$124.62 = +1 \cdot (1.2462) \cdot 10^2$$

with the sign $\sigma = +1$, the exponent $e = 2$, and the significant $\bar{x} = 1.2462$.

- The above example is a five-digit decimal floating-point arithmetic.
- The last digit may need to be changed by rounding.

## Floating-Point Representation in the Binary system

$$x = \sigma \cdot \bar{x} \cdot 10^{e}$$

where $\sigma = +1$ or $-1$ (sign), $e$ is an integer (exponent), and $(1)_2 \le \bar{x} < (10)_2$ is a binary fraction.

For an example,

$$(11011.0111)_2 = (1.10110111)_2 \cdot 2^{(100)_2} \quad \text{with } \sigma = +1,\ e = (100)_2 = 4, \text{ and } \bar{x} = (1.10110111)_2.$$

- The allowable number of binary digits in $\bar{x}$ is called the precision of the binary floating-point representation.

**Single Precision Floating-Point Representation of $x$** has a precision of 24 binary digits and uses 4 bytes (32 bits):

$$x = \sigma \cdot (1.b_{10}\ b_{11}\ \cdots\ b_{32}) \cdot 2^{e} \qquad (\underline{\text{mantissa of decimal digits is 7 or 8}}) \text{ in normalized format } \bar{x}$$

with the exponent $e$ limited by

$$-126 = -(1111110)_2 \le e \le (1111111)_2 = 127.$$

| $\sigma$ | $E$ | $\bar{x}$ |
|---|---|---|
| $b_1$ | $b_2\ b_3\ \cdots\ b_9$ | $b_{10}\ b_{11}\ \cdots\ b_{32}$ |

$$-126 \le e(:= E - 127) \le 127 \quad \text{with} \quad 0 \le E = (b_2\, b_3\, \cdots\, b_9)_2 \le 255$$

- But, if $E = (00\cdots 0)_2 = 0$, then $e = -126$ and $\bar{x} = (0.b_{10}\ b_{11}\ \cdots\ b_{32})_2$ <u>with unnormalized format $\bar{x}$</u>
- if $E = (11\cdots 1)_2 = 255$ and $b_{10} = \cdots = b_{32} = 0$, then $\bar{x} = \pm\infty$,
- if $E = (11\cdots 1)_2 = 255$ and $\sim\big(b_{10} = \cdots = b_{32} = 0\,\big)$ then $\bar{x} = NaN$.

**Double Precision Floating-Point Representation of $x$** has a precision of 53 binary digits and uses 8 bytes (64 bits):

$$\bar{x} = \sigma \cdot (1.b_{13}\ b_{14}\ \cdots\ b_{64}) \cdot 2^e \qquad \text{with } -1022 \le e \le 1023 \qquad (\underline{\text{mantissa of decimal digits is 15 or 16}})$$

| $\sigma$ | $E(:= e + 1023)$ | $\bar{x}$ |
|---|---|---|
| $b_1$ | $b_2\ b_3\ \cdots\ b_{12}$ | $b_{13}\ b_{14}\ \cdots\ b_{64}$ |

## 2.2 Accuracy of Floating-Point Representation

**Machine epsilon** is the difference between 1 and the next larger number that can be stored in the floating-point format.

In single precision IEEE format, the next larger binary number is

$$1.00000000000000000000001$$

with the final binary digit 1 in position 23 to the right of the binary point.

The machine epsilon in single precision format is $2^{-23} \approx 1.19^{-7}$.

In a similar fashion,

The machine epsilon in double precision format is $2^{-52} \approx 2.22^{-16}$.

- In Matlab, it uses the double precision format so that the machine epsilon is eps $\approx 2.22^{-16}$.

**Largest integer $M$** that any integer $x$ $(0 \leq x \leq M)$ can be stored or represented exactly in floating-point form :

- In the single precision format (24 binary digits) :

$$M = (1.00 \cdots 0)_2 \cdot 2^{24} = 2^{24} = 16777216 \approx 1.67^7.$$

- In the double precision format (53 binary digits) :

$$M = (1.00 \cdots 0)_2 \cdot 2^{53} = 2^{53} \approx 9.0^{15}.$$

### 2.2.1 Rounding and Chopping

Let the significant in the floating-point representation contain $n$ binary digits.

If the number $x$ has a significant $\bar{x}$ that requires more than $n$ binary bits, then it must be shortened when $x$ is stored in the computer.

- The simplest method is to simply truncate or chop $\bar{x}$ to $n$ binary digits ignoring the remaining digits.
- The second method is to round $\bar{x}$ to $n$ digits based on the size of the part of $\bar{x}$ following digit $n$.

  Denote the machine floating-point version of a number $x$ by $\mathrm{fl}(x)$.

  Then $\mathrm{fl}(x)$ can be written in the form

$$\mathrm{fl}(x) = x \cdot (1 + \epsilon) \quad \text{with a small number } \epsilon.$$

$$\text{Chopping}: \; -2^{-n+1} \leq \epsilon \leq 0$$

Rounding : $-2^{-n} \leq \epsilon \leq 2^{-n}$ : much better

- The IEEE standard is using the rounding :

Single precision : $-2^{-24} \leq \epsilon \leq 2^{-24}$

Double precision : $-2^{-53} \leq \epsilon \leq 2^{-53}$

### 2.2.2 Errors

Denote by $x_T$ the true value and $x_A$ an approximate value.

Error : $\text{Error}(x_A) = x_T - x_A$

Relative Error : $\text{Rel}(x_A) = \dfrac{x_T - x_A}{x_T}$

### 2.2.3 Sources of Error

Modelling Errors

Blunders and Mistakes

Physical Measurement Errors

Machine Representation and Arithmetic Errors

Mathematical Approximation Errors

## 2.2.4 Loss-of-Significance Errors

Compare the followings :

$$f(x) = x\left(\sqrt{x+1} - \sqrt{x}\right) \qquad f(x) = \frac{x}{\sqrt{x+1} + \sqrt{x}}$$

To avoid the loss of significant digits, use another formulation for $f(x)$, avoiding the subtraction of nearly equal quantities.

## 2.2.5 Noise in Function Evaluation

Using floating-point arithmetic with rounding or chopping, arithmetic operations (e.g., additions and multiplications) cause errors in the evaluation of $f(x)$, generally quite small ones.

## 2.3 Underflow and Overflow Errors

Underflow : Attempts to create numbers that are too small lead to what are called underflow errors.

For example, consider an evaluation

$$f(x) = x^{10} \quad \text{for } x \text{ near } 0.$$

In the single precision arithmetic, the smallest nonzero positive number expressible in *normalized* floating-point format (using the form of significant $\bar{x} = (1.a_1 \cdots a_{23})_2$) is

$$m = (1.0 \cdots 0)_2 \cdot 2^{-126} = 2^{-126} \approx 1.18 \times 10^{-38}.$$

Thus $f(x)$ will be set to zero if

$$x^{10} < m \quad \Longleftrightarrow \quad |x| < \sqrt[10]{m} \approx 1.61 \times 10^{-4} \quad \Longleftrightarrow \quad -0.000161 < x < 0.000161$$

- If the use of *unnormalized* floating-point numbers (using the form of significant $\bar{x} = (0.a_1 \cdots a_{23})_2$) allows to represent the smaller number.

$$m = (0.0 \cdots 1)_2 \cdot 2^{-126} = 2^{-149} \approx 1.4 \times 10^{-45}.$$

- Matlab uses the double precision unnormalized floating-point numbers.

  (Using the form of significant $\bar{x} = (0.a_1 \cdots a_{52})_2$)

$$m = (0.0 \cdots 1)_2 \cdot 2^{-1022} = 2^{-1074} \approx 4.94 \times 10^{-324} \qquad \text{but} \qquad 2^{-1075} = 0 = 10^{-324}.$$

**Overflow** : Attempts to create numbers that are too large lead to what are called overflow errors (more fatal errors).

In the double precision arithmetic, the largest positive number expressible in floating-point format

$$M = (1.1\cdots1)_2 \cdot 2^{1023} = (1.1\cdots1)_2 \cdot 2^{52} \cdot 2^{971}$$

$$= (11\cdots1)_2 \cdot 2^{971} = (2^{53}-1)\cdot 2^{971} \approx 1.80\times 10^{308}.$$

It is possible to eliminate an overflow error by just reformulating the expression being evaluated.

For example, with very large $x$ and $y$ (e.g., $x = 10^{200}$ and $y = 10^{150}$), compare the followings

$$z = \sqrt{x^2+y^2} \qquad \text{and} \qquad z = \begin{cases} |x|\sqrt{1+(y/x)^2}, & 0 \le |y| \le |x| \\ |y|\sqrt{1+(x/y)^2}, & 0 \le |x| \le |y|. \end{cases}$$

**Summation** Add $S$ from smallest to largest terms for the sum

$$S = a_1 + a_2 + \cdots + a_n = \sum_{j=1}^{n} a_j$$

**A Loop Error** Keep away from successive repeated rounding errors in operations.

The following loop

```
for j = 1:n
    x = a + j*h
end
```

is better than

```
x = a;
for j = 1:n
    x = x + h
end
```

in general.

# 3 RootFinding

**Calculating the roots of an equation $f(x) = 0$**

## 3.1 The Bisection Method

Suppose that $f(x)$ is continuous on an interval $a \le x \le b$ and that

$$f(a)f(b) < 0.$$

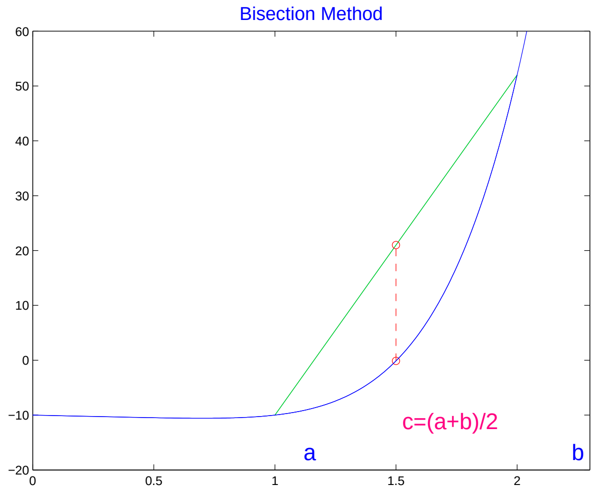
Algorithm (Bisection Method) : To find a root to $f(x) = 0$

Input Arguments : function $f$, Initial guess $a$ and $b$, Tolerance $tol$

Output Argument : approximated root $c$

1. Define $c = (a + b)/2$.
2. If $b - c \le tol$, then accept $c$ as the root and stop.
3. If $sign[f(b)] \cdot sign[f(c)] \le 0$, then set $a = c$.

   Otherwise, set $b = c$.
4. Return to step 1.



## Error Bounds

Let $a_n$, $b_n$ and $c_n$ denote the $n$th computed values of $a$, $b$ and $c$, respectively. Then

$$b_{n+1} - a_{n+1} = \frac{1}{2}(b_n - a_n) = \frac{1}{2^n}(b - a), \quad n \geq 1$$

Since a root $\alpha$ is in either the interval $[a_n, c_n]$ or $[c_n, b_n]$,

$$|\alpha - c_n| \leq c_n - a_n = b_n - c_n = \tfrac{1}{2}(b_n - a_n) = \tfrac{1}{2^n}(b - a). \quad \Leftarrow \text{ linear convergence}$$

Hence,

the iterates $c_n$ converge to $\alpha$ as $n \to \infty$.

## How many iterations?

From

$$|\alpha - c_n| \leq \frac{1}{2^n}(b - a) \leq \epsilon,$$

we have

$$n \geq \frac{\log\left(\frac{b-a}{\epsilon}\right)}{\log 2}.$$

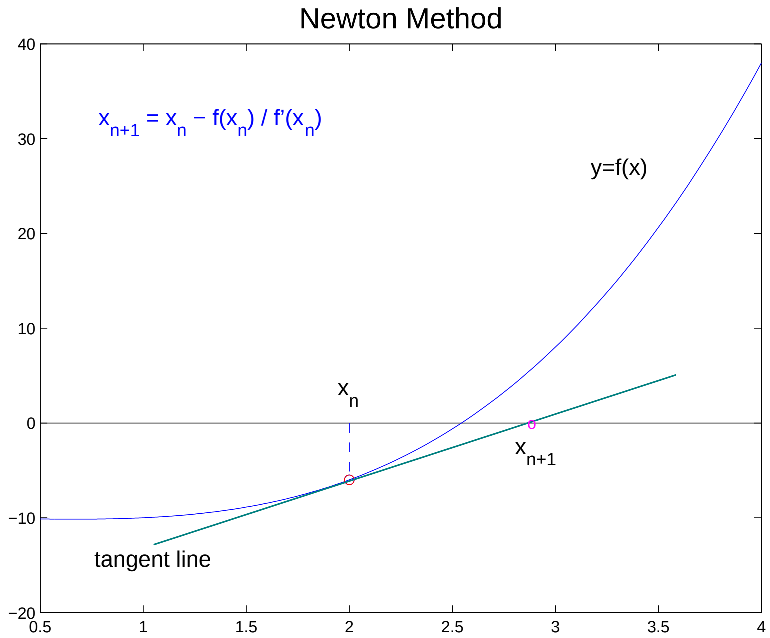
## Advantages

- This method guarantees to converge.
- This method generally converges more slowly than most other methods (e.g., for smooth functions).

Matlab Code : **(Bisection Method)**

```
%--------------------------------------------------%
function [rt, err] = bisect(a,b,tol,maxitr)
%--------------------------------------------------%
  % function [rt, err] = bisect(a,b,tol,maxitr)
  if sign(f(a))*sign(f(b)) > 0
     disp(' f(a) and f(b) are of the same sign.  Stop! '); return
  end
  if a >= b
     tmp = b;  b = a;  a = tmp;   % Make a < b
  end
  c = (a+b)/2;   itr = 0;
  fprintf('\n  itr      a     b    root    f(c)   error   \n')
  while (b-c > tol) & (itr < maxitr)
     itr = itr + 1;
     if sign(f(b))*sign(f(c)) <= 0, a = c;
     else b = c;
     end
     c = (a+b)/2;
     fprintf(' %4.0f  %10.7f  %10.7f  %10.7f  %12.4e  %12.4e \n', ...
                 itr, a, b, c, f(c), b-c);
  end
     rt = c;   err = b - c;
%--------------------------------------------------%
function y = f(x)
   y = x.^6 - x - 1;
```

## 3.2 Newton's Method




Let $y = p_1(x)$ be the linear Taylor polynomial (the tangent line passing through $(x_0, f(x_0))$ ) of $y = f(x)$ at $x = x_0$:

$$p_1(x) = f(x_0) + f'(x_0)(x - x_0).$$

If $x_1$ is a root of $p_1(x)$, then

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

**Algorithm (Newton's Iteration)** : To find a root of $f(x)$

With an initial guess $x_0$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2, \ldots.$$

**Error Bounds**

Assume that $f \in C^2$ in some interval about the root $\alpha$ and $f'(\alpha) \neq 0$.

Using Taylor's theorem yields

$$0 = f(\alpha) = f(x_n) + (\alpha - x_n)f'(x_n) + \frac{1}{2}(\alpha - x_n)^2 f''(c_n)$$

with $c_n$ an unknown point between $\alpha$ and $x_n$.

$$0 = \frac{f(x_n)}{f'(x_n)} + \alpha - x_n + (\alpha - x_n)^2 \frac{f''(c_n)}{2f'(x_n)}$$

Using Newton's iteration $\dfrac{f(x_n)}{f'(x_n)} = x_n - x_{n+1}$,

$$0 = x_n - x_{n+1} + \alpha - x_n + (\alpha - x_n)^2 \frac{f''(c_n)}{2f'(x_n)} \quad \text{or} \quad \alpha - x_{n+1} = (\alpha - x_n)^2 \left[ -\frac{f''(c_n)}{2f'(x_n)} \right].$$

Thus, we have the second order error estimates (quadratic convergence):

$$M|\alpha - x_n| \le |M(\alpha - x_{n-1})|^2 = \cdots = |M(\alpha - x_0)|^{2^n} \quad \text{where} \quad \left| \frac{f''(c_n)}{2f'(x_n)} \right| \le M.$$

- If the initial error is sufficiently small, then the error in the succeeding iterate will decrease very rapidly.

**Example.** Find a root $f(x) = x^6 - x - 1$ with an initial guess $x_0 = 1.5$ and then compare this with the results for the bisection method.

**Example (How to compute $\frac{1}{b}$ without the division?).**

Assume $b > 0$. Let $f(x) = b - \dfrac{1}{x}$. Then the root is $\alpha = \dfrac{1}{b}$.

Using the derivative $f'(x) = \dfrac{1}{x^2}$, we have the Newton Method given by

$$x_{n+1} = x_n - \frac{b - \frac{1}{x_n}}{\frac{1}{x_n^2}} \quad \text{or} \quad x_{n+1} = x_n(2 - bx_n).$$

This involves only multiplication and subtraction.

For the error, using $\alpha = \dfrac{1}{b}$ yields

$$Rel(x_n) = [Rel(x_{n-1})]^2 = \cdots = [Rel(x_0)]^{2^n} \qquad (n \geq 0).$$

The Newton iteration converges to $\alpha = \dfrac{1}{b}$ with the second order if and only if

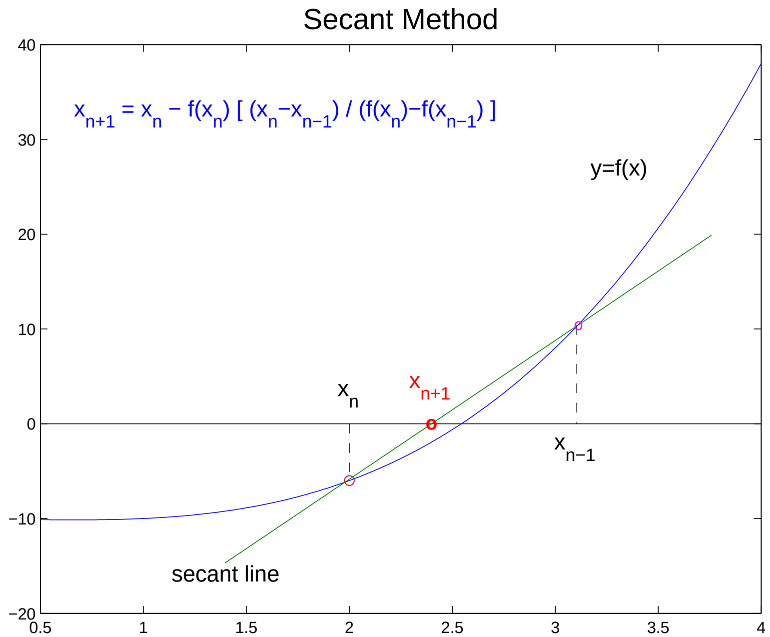
$$|Rel(x_0)| < 1 \iff -1 < \frac{\frac{1}{b} - x_0}{\frac{1}{b}} < 1 \iff 0 < x_0 < \frac{2}{b}.$$

If $|Rel(x_0)| = 0.1$, then $|Rel(x_n)| = 10^{-2^n}$. (e.g., $|Rel(x_4)| = 10^{-16}$)

**What is the first order error?**

$$|e_n| = \gamma\, |e_{n-1}| = \cdots = \gamma^n\, |e_0| \qquad \text{with some } 0 < \gamma < 1.$$

# 3.3 Secant Method




Let $y = p(x)$ be the linear polynomial (the secant line) passing through $(x_0, f(x_0))$ and $(x_1, f(x_1))$ :

$$p(x) = f(x_1) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_1).$$

If $x_2$ is the root of $p(x)$, then $x_2 = x_1 - f(x_1)\dfrac{x_1 - x_0}{f(x_1 - f(x_0)}$.

**Algorithm (Secant Method)** : To find a root of $f(x)$

With two initial guesses $x_0$ and $x_1$

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}, \quad n \geq 1.$$

## Lagrange Interpolation Polynomial Approximation

If $f \in C^{n+1}$ in some interval $[a, b]$ and $\{x_k\}_{k=0}^n \subset [a, b]$, then there exists a number $\xi(x) \in (a, b)$ such thaat

$$f(x) = P_n(x) + \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0) \cdots (x - x_n)$$

with the Lagrange interpolation and the $k$-th Lagrange shape polynomial of degree $n$ :

$$P_n(x) = \sum_{k=0}^{n} f(x_k) L_{n,k}(x) \quad \text{and} \quad L_{n,k}(x) = \prod_{i=0, i \neq k}^{n} \frac{(x - x_i)}{(x_k - x_i)} = \frac{(x - x_0)}{(x_k - x_0)} \cdots \frac{(x - x_n)}{(x_k - x_n)}.$$

## Error Bounds

Assume that $f \in C^2$ in some interval about the root $\alpha$ and $f'(\alpha) \neq 0$.

Using Lagrange interpolation yields

$$f(x) = \frac{x - x_{n-1}}{x_n - x_{n-1}} f(x_n) + \frac{x - x_n}{x_{n-1} - x_n} f(x_{n-1}) + \frac{1}{2}(x - x_{n-1})(x - x_n) f''(\xi)$$

and then

$$0 = f(\alpha) = \frac{\alpha - x_{n-1}}{x_n - x_{n-1}} f(x_n) + \frac{\alpha - x_n}{x_{n-1} - x_n} f(x_{n-1}) + \frac{1}{2}(\alpha - x_{n-1})(\alpha - x_n) f''(\xi_n)$$

with $\xi_n$ an unknown point between $\min\{\alpha, x_{n-1}, x_n\}$ and $\max\{\alpha, x_{n-1}, x_n\}$.

Also, we have from the Mean Value theorem that

$$f(x_{n-1}) = f(x_n) - (x_n - x_{n-1}) f'(\zeta_n)$$

with $\zeta_n$ an unknown point between $x_{n-1}$ and $x_n$.

From the last two equations, we have

$$0 = f(x_n) + (\alpha - x_n)f'(\zeta_n) + \frac{1}{2}(\alpha - x_{n-1})(\alpha - x_n)f''(\xi_n)$$

or

$$0 = \frac{f(x_n)}{f'(\zeta_n)} + (\alpha - x_n) + (\alpha - x_{n-1})(\alpha - x_n)\frac{f''(\xi_n)}{2f'(\zeta_n)}.$$

Now, using the secant iteration

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} = x_n - \frac{f(x_n)}{f'(\zeta_n)},$$

$$0 = (\alpha - x_{n+1}) + (\alpha - x_{n-1})(\alpha - x_n)\frac{f''(\xi_n)}{2f'(\zeta_n)}$$

so that

$$\alpha - x_{n+1} = (\alpha - x_{n-1})(\alpha - x_n)\left[\frac{-f''(\xi_n)}{2f'(\zeta_n)}\right].$$

Let us consider the following identity

$$|e_{n+1}| = |e_n| \cdot |e_{n-1}|\, M \quad \text{with} \quad M \approx \left|\frac{-f''(\alpha)}{2f'(\alpha)}\right| \approx \left|\frac{-f''(\xi_n)}{2f'(\zeta_n)}\right|, \quad \forall\, n.$$

Then

$$\frac{|e_{n+1}|}{|e_n|^\gamma} = M \cdot \left(\frac{|e_n|}{|e_{n-1}|^\gamma}\right)^\alpha \quad \text{with} \quad \gamma = \frac{1+\sqrt{5}}{2}, \quad \alpha = \frac{1-\sqrt{5}}{2}, \ (i.e., \alpha + \gamma = 1,\ \alpha\gamma = -1)$$

so that

$$M^\beta \frac{|e_{n+1}|}{|e_n|^\gamma} = \left(M^\beta \frac{|e_n|}{|e_{n-1}|^\gamma}\right)^\alpha = \left(M^\beta \frac{|e_1|}{|e_0|^\gamma}\right)^{\alpha^n} \quad \text{with} \quad \beta = \frac{1}{\alpha - 1}.$$

Taking the limit

$$\lim_{n\to\infty}\frac{|e_{n+1}|}{|e_n|^\gamma} = M^{-\beta}\lim_{n\to\infty}\left(M^\beta\frac{|e_1|}{|e_0|^\gamma}\right)^{\alpha^n} = M^{-\beta} = M^{\gamma-1} \quad \text{because} \quad \lim_{n\to\infty}\alpha^n = 0,$$

we have the following approximate error estimates:

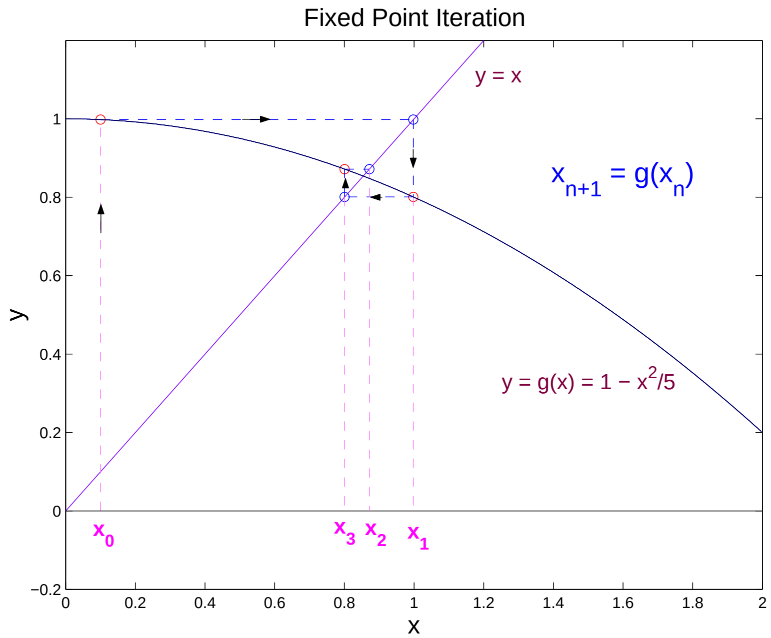
$$\boxed{|e_{n+1}| = |\alpha - x_{n+1}| \approx c\,|\alpha - x_n|^\gamma = c\,|e_n|^\gamma \quad \text{with} \quad \gamma = \frac{1+\sqrt{5}}{2} \approx 1.62.}$$

- If the initial error is sufficiently small, then the error in the succeeding iterate will decrease very rapidly.
- Newton method converges more rapidly than the secant method but it requires two function evaluations per iteration, that of $f(x_n)$ and $f'(x_n)$.
- And the secant method requires only one evaluation $f(x_n)$.
- For many problems with a complicated $f'(x)$, the secant method will probably be faster in actual running time on a computer.

**Example.** Find a root $f(x) = x^6 - x - 1$ with an initial guess $x_0 = 1.5$ and then compare this with the results for the Newton method.

# 3.4 Fixed Point Iteration

**To find a fixed-point $\alpha$ of $g(x)$ : $\alpha = g(\alpha)$**




**Algorithm (Fixed-Point Iteration)** : To find a root $\alpha$ of $x = g(x)$

With an initial guess $x_0$

$$x_{n+1} = g(x_n), \quad n \geq 0.$$

- If the sequence $x_n$ converges, then $\lim_{n\to\infty} x_n = \alpha$.

**Lemma 3.1.** *Let $g(x)$ be a continuous function and suppose $g$ satisfies*

$$a \leq g(x) \leq \quad \textit{for} \quad a \leq x \leq b.$$

*Then, the equation $x = g(x)$ has at least one solution $\alpha \in [a, b]$.*

**Theorem 3.2 (Contraction Mapping Theorem).** *Assume $g(x)$ and $g'(x)$ are continuous on $[a, b]$, and assume $a \leq g(x) \leq b$ for $a \leq x \leq b$. Further assume that*

$$\lambda := \max_{a \leq x \leq b} |g'(x)| < 1.$$

*Then*

1. *There is a unique solution $\alpha$ of $x = g(x)$ in the interval $[a, b]$.*
2. *For any initial guess $x_0 \in [a, b]$, $x_n$ converges to $\alpha$.*
3. $|\alpha - x_n| \leq \dfrac{\lambda^n}{1-\lambda} |x_1 - x_0|, \quad (n \geq 0).$
4. $\displaystyle\lim_{n \to \infty} \frac{\alpha - x_{n+1}}{\alpha - x_n} = g'(\alpha) \quad$ *so that* $\quad \alpha - x_{n+1} \approx g'(\alpha)(\alpha - x_n).$

**Corollary 3.3.** *Assume $\alpha \in (c, d)$. If $g(x)$ and $g'(x)$ are continuous in $(c, d)$, and if*

$$|g'(\alpha)| < 1,$$

*then, there is an interval $[a, b]$ around $\alpha$ for which the conclusions of the above theorem are true.*

## Error Bounds

We have the linear convergence error bound :

$$\boxed{|\alpha - x_{n+1}| = \lambda\,|\alpha - x_n| \quad \text{so that} \quad |\alpha - x_n| = \lambda^n\,|\alpha - x_0|.}$$

## Aitken Error Estimation

: to estimate the error

Denote by $\lambda = g'(\alpha)$. Then

$$\alpha - x_n \approx \lambda(\alpha - x_{n-1})$$

or

$$\alpha \approx x_n + \frac{\lambda}{1-\lambda}(x_n - x_{n-1})$$

Denote by

$$\lambda_n := \frac{x_n - x_{n-1}}{x_{n-1} - x_{n-2}}.$$

Then we have the Aitken's extrapolation formula :

$$\alpha - x_n \approx \frac{\lambda_n}{1-\lambda_n}(x_n - x_{n-1}).$$