Numerical Analysis
- Advanced Topics in Root Finding -

Hanyang University

Jong-Il Park

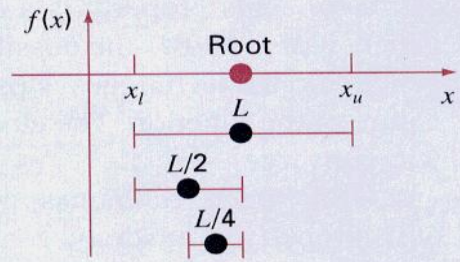
Summary: Root Finding

Bracketing methods:

Bisection

$$x_r = \frac{x_l + x_u}{2}$$

$$\text{If } f(x_l)f(x_r) < 0,\ x_u = x_r$$
$$f(x_l)f(x_r) > 0,\ x_l = x_r$$



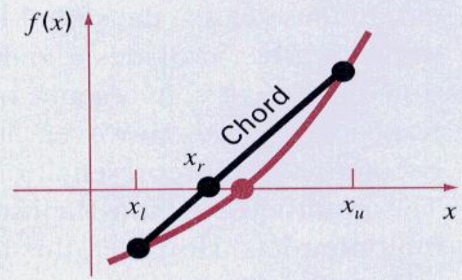
Stopping criterion:

$$\left|\frac{x_r^{new} - x_r^{old}}{x_r^{new}}\right| 100\% \le \epsilon_s$$

False position

$$x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$$

$$\text{If } f(x_l)f(x_r) < 0,\ x_u = x_r$$
$$f(x_l)f(x_r) > 0,\ x_l = x_r$$

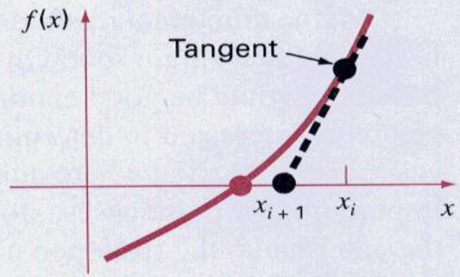


Stopping criterion:

$$\left|\frac{x_r^{new} - x_r^{old}}{x_r^{new}}\right| 100\% \le \epsilon_s$$

Newton-Raphson

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$



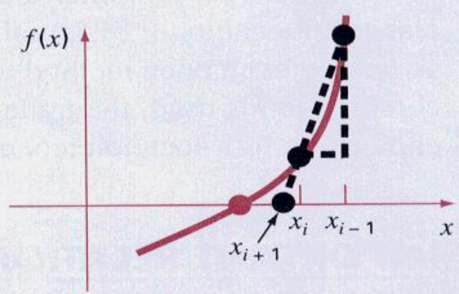
Stopping criterion:

$$\left|\frac{x_{i+1} - x_i}{x_{i+1}}\right| 100\% \le \epsilon_s$$

Error: $E_{i+1} = O(E_i^2)$

Secant

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$



Stopping criterion:

$$\left|\frac{x_{i+1} - x_i}{x_{i+1}}\right| 100\% \le \epsilon_s$$

Error analysis of N-R method

Taylor series: $$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + \frac{f''(\xi)}{2!}(x_{i+1} - x_i)^2$$

Newton-Raphson method: $$0 = f(x_i) + f'(x_i)(x_{i+1} - x_i) \quad \text{--- (1)}$$

At the true solution x_r : $$0 = f(x_i) + f'(x_i)(x_r - x_i) + \frac{f''(\xi)}{2!}(x_r - x_i)^2 \quad \text{--- (2)}$$

(1) → (2) :

$$0 = f'(x_i)(x_r - x_{i+1}) + \frac{f''(\xi)}{2!}(x_r - x_i)^2$$

Let $E_{t,i+1} = x_r - x_{i+1}$

$$\boxed{E_{t,i+1} = \frac{-f''(x_r)}{2f'(x_r)} E_{t,i}^2}$$

Quadratic convergence!

Error Analysis of Secant Method

- Convergence [Jeeves, 1958]

$$e_{k+1} = \left\{ \frac{1}{2} \frac{f'(\alpha)}{f(\alpha)} \right\}^{0.618} e_k^{1.618}$$

  - More efficient than N-R method if the calculation of *f'(x)* is complex

- Modified secant method

$$f'(x_i) \cong \frac{f(x_i + \delta x_i) - f(x_i)}{\delta x_i}$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \Rightarrow \boxed{x_{i+1} = x_i - \frac{\delta x_i f(x_i)}{f(x_i + \delta x_i) - f(x_i)}}$$

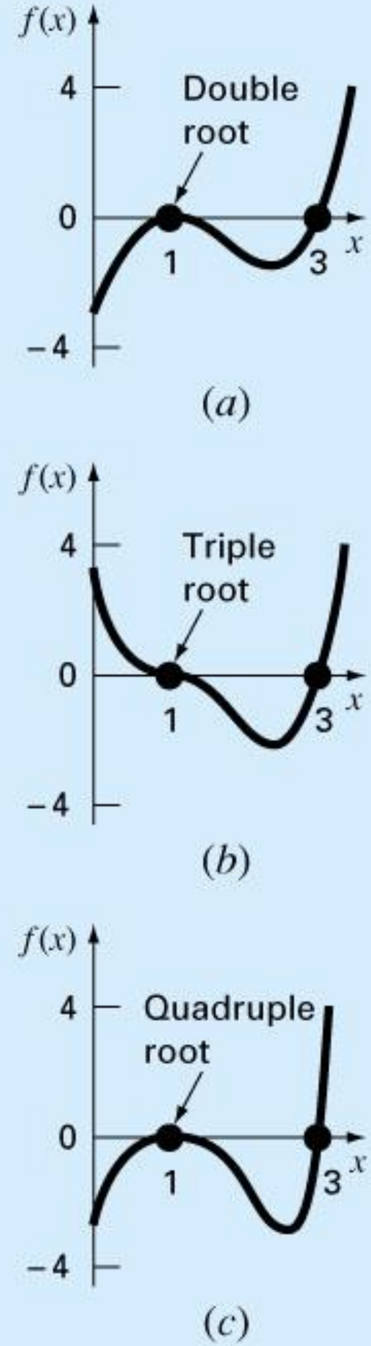
Multiple roots

- Bracketing methods cannot cope with multiple roots
- Open methods can find multiple roots
  - But the speed is slow in many cases
  - f'(x)→0 will cause a problem(divide by zero)

    f(x) will always reach zero before f'(x) [Ralston and Rabinowitz, 1978]

    ➔ if a zero check for f(x) is incorporated into the program, the computation can be terminated before f'(x) reaches zero
  - Alternative way using u(x)=f(x)/f'(x)

$$x_{i+1} = x_i - \frac{f(x_i)\,f'(x_i)}{[f'(x_i)]^2 - f(x_i)\,f''(x_i)}$$



Polynomial evaluation

- Bad method

```
p=c[0]+c[1]*x+c[2]*x*x+c[3]*x*x*x+c[4]*x*x*x*x;
```

- Worst method

```
p=c[0]+c[1]*x+c[2]*pow(x,2.0)+c[3]*pow(x,3.0)+c[4]*pow(x,4.0);
```

- Best method

```
p=c[0]+x*(c[1]+x*(c[2]+x*(c[3]+x*c[4])));
```

or

```
p=(((c[4]*x+c[3])*x+c[2])*x+c[1])*x+c[0];
```

C code

```
p=c[n];
for(j=n-1;j>=0;j--) p=p*x+c[j];
```

or

```
p=c[j=n];
while (j>0) p=p*x+c[--j];
```

Polynomial differentiation

```
p=c[n];
dp=0.0;
for(j=n-1;j>=0;j--) {dp=dp*x+p; p=p*x+c[j];}
```

or

```
p=c[j=n];
dp=0.0;
while (j>0) {dp=dp*x+p; p=p*x+c[--j];}
```

N-th derivatives

```
void ddpoly(float c[], int nc, float x, float pd[], int nd)
```

Given the nc+1 coefficients of a polynomial of degree nc as an array c[0..nc] with c[0] being the constant term, and given a value x, and given a value nd>1, this routine returns the polynomial evaluated at x as pd[0] and nd derivatives as pd[1..nd].

```
{
    int nnd,j,i;
    float cnst=1.0;

    pd[0]=c[nc];
    for (j=1;j<=nd;j++) pd[j]=0.0;
    for (i=nc-1;i>=0;i--) {
        nnd=(nd < (nc-i) ? nd : nc-i);
        for (j=nnd;j>=1;j--)
            pd[j]=pd[j]*x+pd[j-1];
        pd[0]=pd[0]*x+c[i];
    }
    for (i=2;i<=nd;i++) {        After the first derivative, factorial constants come in.
        cnst *= i;
        pd[i] *= cnst;
    }
}
```

Polynomial deflation

- Multiplication by (x-a)

```
c[n]=c[n-1];
for (j=n-1;j>=1;j--) c[j]=c[j-1]-c[j]*a;
c[0] *= (-a);
```

- Synthetic division by (x-a)

```
rem=c[n];
c[n]=0.0;
for(i=n-1;i>=0;i--) {
    swap=c[i];
    c[i]=rem;
    rem=swap+rem*a;
}
```

$$(2x^4 + 3x^2 + 5x + 1) \div (x - 2)$$

Bairstow's method

- Deflation method

$$f_n(x) = a_0 + a_1 x + \cdots + a_n x^n$$

$$f_n(x) = (x^2 - rx - s)(b_2 + b_3 x + \cdots + b_n x^{n-2}) + b_1(x - r) + b_0$$

  - Find r and s such that $b_0 = b_1 = 0$
  - Efficient routine using synthetic division exists
  - Good initial guess of r, s is important
  - Complex roots can be evaluated
  - Suitable for root polishing

- In Numerical Recipes in C

  void qroot();

➔ Read Sect.7.5.

Basic Concept

- Determining a root of a polynomial: divide the polynomial by the factor $x - t$

$$f_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$
$$= (b_nx^{n-1} + b_{n-1}x^{n-2} + \cdots + b_2x + b_1)(x-t) + b_0$$

$$\downarrow$$

$$f_{n-1}(x) = b_1 + b_2x + b_3x^2 + \cdots + b_nx^{n-1}$$

- With a remainder $R = b_0$, where the coefficients can be calculated by the recurrence relationship

$$b_n = a_n$$
$$b_i = a_i + b_{i+1}t \qquad \text{for } i = n-1 \text{ to } 0$$

▶ **if *t* is a root of the original polynomial, the remainder b_0 is equal to zero**

Basic Concept

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$
$$= (b_n x^{n-1} + b_{n-1} x^{n-2} + \cdots + b_2 x + b_1)(x - t) + b_0$$

$$\downarrow$$

$$\begin{array}{r|l|l|l|l}
b_n x^n & + \quad b_{n-1} x^{n-1} & + \quad \cdots & + \quad b_1 x & \\
 & -t b_n x^{n-1} & - \quad \cdots & -t b_2 x & -t b_1 \\
+ & & & & b_0 \\
\hline
b_n x^n & + (b_{n-1} - t b_n) x^{n-1} & + \quad \cdots & + (b_1 - t b_2) x & + (b_0 - t b_1) \\
a_n x^n & + \quad a_{n-1} x^{n-1} & + \quad \cdots & + \quad a_1 x & + \quad a_0
\end{array}$$

$$\downarrow$$

$$\boxed{\begin{array}{l} b_n = a_n \\ b_i = a_i + b_{i+1} t \qquad \text{for } i = n-1 \text{ to } 0 \end{array}}$$

Evaluation of complex roots

- Dividing polynomial by a quadratic factor $x^2 - rx - s$

$$f_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

$$\downarrow$$

$$f_{n-2}(x) = b_2 + b_3 x + \cdots + b_{n-1} x^{n-3} + b_n x^{n-2}$$

- with a remainder $R = b_1(x - r) + b_0$ as with normal synthetic division, a simple recurrence relationship can be used to perform the division by the quadratic factor:

$$\begin{aligned} b_n &= a_n \\ b_{n-1} &= a_{n-1} + r b_n \\ b_i &= a_i + r b_{i+1} + s b_{i+2} \qquad \text{for } i = n-2 \text{ to } 0 \end{aligned}$$

Taylor series

- Because both b_0 and b_1 are functions of both r and s, they can be expanded using a Taylor series

$$b_1(r+\Delta r, s+\Delta s) = b_1 + \frac{\partial b_1}{\partial r}\Delta r + \frac{\partial b_1}{\partial s}\Delta s$$

$$b_0(r+\Delta r, s+\Delta s) = b_0 + \frac{\partial b_0}{\partial r}\Delta r + \frac{\partial b_0}{\partial s}\Delta s$$

equal to zero to give

$$\frac{\partial b_1}{\partial r}\Delta r + \frac{\partial b_1}{\partial s}\Delta s = -b_1$$

$$\frac{\partial b_0}{\partial r}\Delta r + \frac{\partial b_0}{\partial s}\Delta s = -b_0$$

$$c_n = b_n$$
$$c_{n-1} = b_{n-1} + r c_n$$
$$c_i = b_i + r c_{i+1} + s c_{i+2}$$
for $i = n-2$ to 1

$$\partial b_0/\partial r = c_1$$
$$\partial b_0/\partial s = \partial b_1/\partial r = c_2$$
$$\partial b_1/\partial s = c_3$$

$$c_2 \Delta r + c_3 \Delta s = -b_1$$
$$c_1 \Delta r + c_2 \Delta s = -b_0$$

▶ **These equations can be solved for Δr and Δs , which can in turn be employed to improve the initial guesses of r and s !**

Taylor series

- At each step, an approximate error in r and s can be estimated, as in

$$|\varepsilon_{a,r}| = \left|\frac{\Delta r}{r}\right| 100\% \qquad \text{and} \qquad |\varepsilon_{a,s}| = \left|\frac{\Delta s}{s}\right| 100\%$$

- When both of these error estimates fall below a prespecified stopping criterion ε_s, the values of the roots can be determined by

$$x = \frac{r \pm \sqrt{r^2 + 4s}}{2}$$

- At this point, three possibilities exist:
  1. The quotient is a third-order polynomial or greater => repeat deflation
  2. The quotient is a quadratic, => obtain roots
  3. The quotient is a first-order polynomial

Laguerre method

- Deflation method
- Algorithm derivation

$$P(x) = (x - x_1)(x - x_2)\cdots(x - x_n)$$

$$\ln|P(x)| = \ln|x - x_1| + \ln|x - x_2| + \cdots + \ln|x - x_n|$$

$$\frac{d\ln|P(x)|}{dx} = \frac{1}{x - x_1} + \frac{1}{x - x_2} + \cdots + \frac{1}{x - x_n}$$

$$= \frac{P'}{P} \equiv G$$

$$-\frac{d^2\ln|P(x)|}{dx^2} = \frac{1}{(x - x_1)^2} + \frac{1}{(x - x_2)^2} + \cdots + \frac{1}{(x - x_n)^2}$$

$$= \left[\frac{P'}{P}\right]^2 - \frac{P''}{P} \equiv H$$

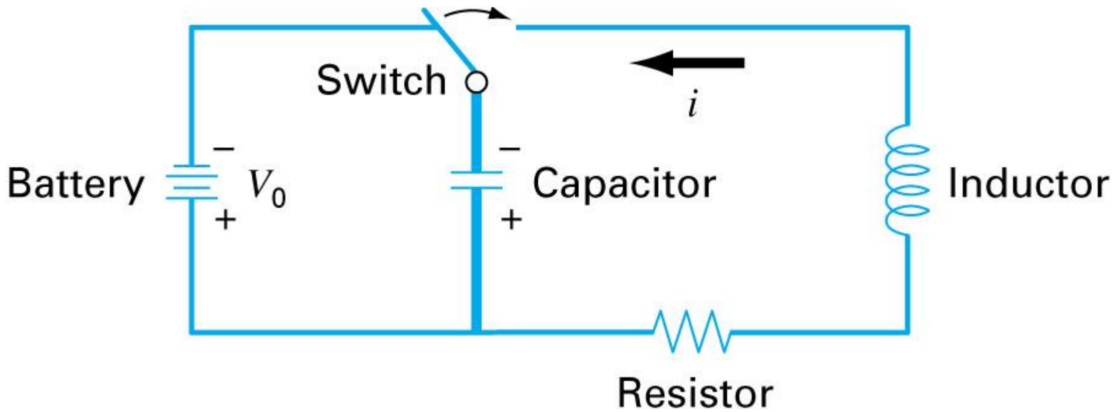
Let $x - x_1 = a \quad ; \quad x - x_i = b, \quad i = 2, 3, \cdots, n$

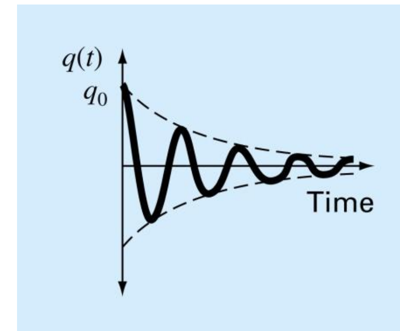
Then

$$a = \frac{n}{G \pm \sqrt{(n-1)(nH - G^2)}}$$

- In Numerical Recipes in C: zroots() calls laguer();

Application of Root Finding:
Electric circuit design(1/2)





- Problem: Find the proper R to dissipate energy to 1% at a specified rate(t=0.05s), given L=5H, C=10^{-4} F.
- Solution:

$$f(R) = e^{-Rt/(2L)} \cos\left[\sqrt{\frac{1}{LC} - \left(\frac{R}{2L}\right)^2}\, t\right] - \frac{q}{q_0}$$

Application of Root Finding:

Electric circuit design(2/2)

$$f(R) = e^{-0.005R} \cos\left[\sqrt{2000 - 0.01R^2}\, 0.05\right] - 0.01$$

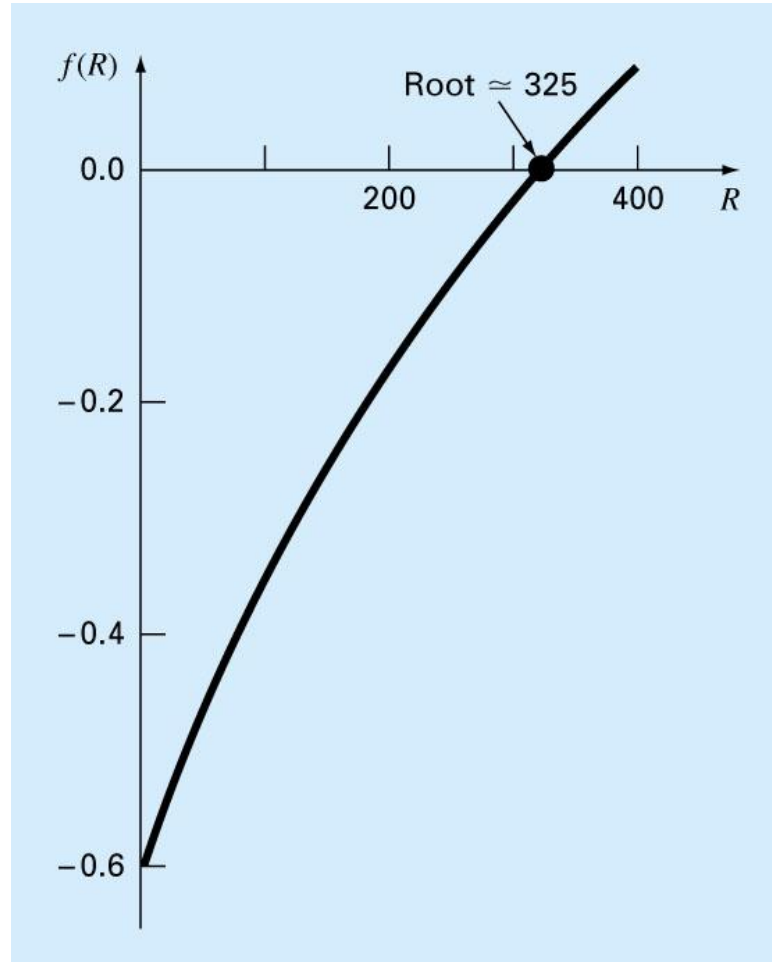
- Reasonable initial range for R:
  0< R < 400

- To achieve r.e. of 10^{-4} %

  Bisection method: 21 iterations
  Other methods: ?

- To achieve r.e. of 10^{-6} %

  Bisection method: ? iterations
  Other methods: ?
  [Homework]



Homework

- Find the root of f(R)=0 and the number of iterations when the r.e.=10^{-4} and 10^{-6} respectively.

$$f\ (R) = e^{-0.005R}\cos\left[\sqrt{2000-0.01R^2}\,0.05\right]-0.01$$

- Solve the problems: 8-32, 36