OLD DOMINION UNIVERSITY

Nonlinear Equations

A. Godunov

1. Prelude
2. Closed domain methods
3. Open domain methods
4. Roots of polynomials
5. Systems of nonlinear equations

updated 9 February 2022

1

Part 1:

Prelude to the problem

2

Non-linear vs. linear equation.

Very many problems in physics require a solution of non-linear equation or function. And non-linear equations are much harder to solve.

Examples:
A system of linear equations vs. a system of non-linear equations, a linear ordinary differential equation vs. a non-linear ODE, etc.

Very often we try to transform a nonlinear problem to a linear one by using proper approximations.

However, very many nonlinear problems cannot be linearized without losing the essence of physics behind.

3

3

Nonlinear vs. linear equation.

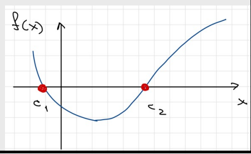
In this lecture we consider as simple as possible form of nonlinear equations, namely a nonlinear function of one variable $f(x) = 0$.

Statement of the problem:

Given the continuous nonlinear function $f(x)$, find the value(s) $x = c$ such that $f(c) = 0$

The non-linear function $f(x)$ can be

- an algebraic equation
- a transcendental equation
- a solution of a differential equation
- ... any non-linear equation



4

Examples

Simple equations of one variable

$$x^2 - 6x + 9 = 0$$
$$x - \cos(x) = 0$$
$$\exp(x)\ln(x^2) - x\cos(x) = 0$$

Quantum mechanics: Solutions of the Schrodinger equation for a finite square well $U(x) = -U_0$ for $|x| < a$, can be found from the following non-linear equations (for even and odd states

$$\sqrt{U_0 - |E|}\tan\sqrt{2m(U_0 - |E|)\,a^2/\hbar^2} = \sqrt{E}$$

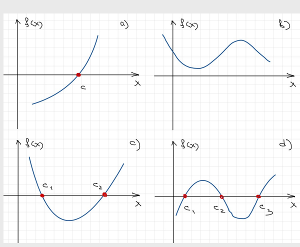
$$\sqrt{U_0 - |E|}\cot\sqrt{2m(U_0 - |E|)\,a^2/\hbar^2} = -\sqrt{E}$$

In textbooks these equations are solved only graphically.

5

5

Solutions of nonlinear functions



a) a single real root

b) no real roots exist (but complex roots may exist)

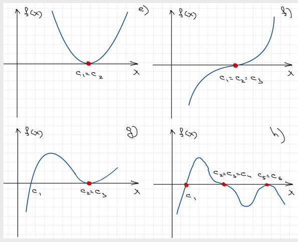
c) two simple roots

d) three simple roots

6

6

Solutions of nonlinear functions (cont.)



e) two multiple roots

f) three multiple roots

g) one simple root and two multiple roots

h) multiple roots

7

7

Prelude for root finding

1. All non-linear equations can only be solved iteratively.
2. We must guess an approximate root to start an iterative procedure.

The better we guess, the more chances we have to find the right root in shorter time.

8

8

Bounding and refining

There are two distinct phases in finding the solution of at nonlinear equation.

1. Bounding the solution
2. Refining the solution

9

9

Phase 1
Bounding the Solution

Bounding the solution involves finding a rough estimate of the solution that can be used as the initial approximation, in an iterative procedure that refines the solution to a specified tolerance.

If possible, the root should be bracketed between two points at which the value of the nonlinear function has opposite signs.

1. Graphing the function
2. Incremental search
3. Past experience with the problem or a similar one
4. Solution of a simplified approximate model
5. Previous solution in a sequence of solutions

"The hardest thing of all is to find a black cat in a dark room, especially if there is no cat." *Confucius (551-479 AD)*

10

10

Phase 2
Iterative Refining the Solution

Iterative refining the solution involves determining the solution to a specified tolerance by a systematic procedure.

There are numerous pitfalls in finding the roots of nonlinear equations.

Two types of methods:

1. Closed domain (bracketing) methods
2. Open domain (non-bracketing) methods

Important question:
How to stop an iteration?

abs. error $\left|g_{i+1} - g_i\right|$

rel. error $\left|\dfrac{g_{i+1} - g_i}{g_{i+1}}\right|$ never use $\left|1 - \dfrac{g_i}{g_{i+1}}\right|$

11

11

Part 2:
Closed domain methods

12

Closed Domain (Bracketing) Methods

Methods start with two values of x which bracket the root in the interval $[a, b]$.

If $f(a)$ and $f(b)$ have opposite signs, and if the function is continuous, then at least one root must be in the interval.

Most common closed domain methods:

- Interval halving (bisection)
- False position (regula falsi)

Bracketing methods are robust
(they are guaranteed to obtain a solution
since the root is trapped in the closed interval).

13

13

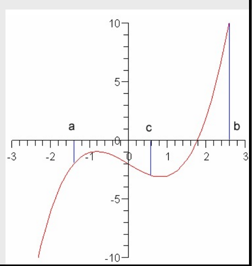
2.1 Bisectional method

The simplest but the most robust method!

1. let $f(x)$ be a continuous function on $[a, b]$
2. let $f(x)$ changes sign between a and b, $f(a)f(b) < 0$

Example: function

$f(x) = x^3 - 2x - 2$



14

Bisectional method - algorithm

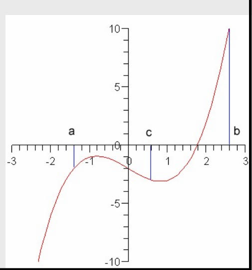
Divide $[a, b]$ into tow equal parts with $c = (a + b)/2$ and if

$$f(a)f(c)\begin{cases} < 0 & \text{then there is a root in } [a, c], set\ a = a, b = c \\ > 0 & \text{then there is a root in } [c, b], set\ a = c, b = b \\ = 0 & \text{then c is the root} \end{cases}$$

Interval halving is an iterative procedure.

The iterations are continued until

$|b_i - a_i| \le \varepsilon_1$ or $|f(c_i)| \le \varepsilon_2$ or both



15

Bisectional method – summary

- The root is bracketed within the bounds of the interval, so the method is guaranteed to converge
- On each bisectional step we reduce by two the interval where the solution occurs. After n steps the original interval $[a, b]$ will be reduced to the $(b-a)/2n$ interval. The bisectional procedure is repeated till $(b-a)/2n$ is less than the given tolerance $(b_n - a_n) < \varepsilon$.
  thus, n is given by

$$n = \frac{1}{\ln 2} \ln\left(\frac{b_0 - a_0}{b_n - a_n}\right)$$

- The major disadvantage of the bisection method is that the solution converges slowly.
- The method does not use information about actual function behavior

16

16

Example: C++

```
 double bisect(double a, double b, double eps)
{
    double xl,x0,xr;

    if( f(a)*f(b) > 0.0) return 999;

    xl = a;
    xr = b;
    while (fabs(xr - xl) >= eps)
    {
      x0 = (xr + xl)/2.0;
      if((f(xl) * f(x0)) <= 0.0 ) xr = x0;
      else xl = x0;
     }
       return x0;
}
```

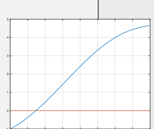
17

17

Example for y = x - cos(x) on [0.0,4.0] for eps = 1.0e-6

i	a	f(a)	b	f(b)	c	f(c)
1	0.00000	-1.00000	4.00000	4.65364	2.00000	2.41615
2	0.00000	-1.00000	2.00000	2.41615	1.00000	0.45970
3	0.00000	-1.00000	1.00000	0.45970	0.50000	-0.37758
4	0.50000	-0.37758	1.00000	0.45970	0.75000	0.01831
5	0.50000	-0.37758	0.75000	0.01831	0.62500	-0.18596
6	0.62500	-0.18596	0.75000	0.01831	0.68750	-0.08533
7	0.68750	-0.08533	0.75000	0.01831	0.71875	-0.03388
8	0.71875	-0.03388	0.75000	0.01831	0.73438	-0.00787
9	0.73438	-0.00787	0.75000	0.01831	0.74219	0.00520
10	0.73438	-0.00787	0.74219	0.00520	0.73828	-0.00135
11	0.73828	-0.00135	0.74219	0.00520	0.74023	0.00192
12	0.73828	-0.00135	0.74023	0.00192	0.73926	0.00029
13	0.73828	-0.00135	0.73926	0.00029	0.73877	-0.00053
14	0.73877	-0.00053	0.73926	0.00029	0.73901	-0.00012
15	0.73901	-0.00012	0.73926	0.00029	0.73914	0.00008
16	0.73901	-0.00012	0.73914	0.00008	0.73907	-0.00002
17	0.73907	-0.00002	0.73914	0.00008	0.73911	0.00003
18	0.73907	-0.00002	0.73911	0.00003	0.73909	0.00001
19	0.73907	-0.00002	0.73909	0.00001	0.73908	-0.00000
20	0.73908	-0.00000	0.73909	0.00001	0.73909	0.00000
21	0.73908	-0.00000	0.73909	0.00000	0.73908	-0.00000
22	0.73908	-0.00000	0.73909	0.00000	0.73909	0.00000

iterations	root
22	0.73909



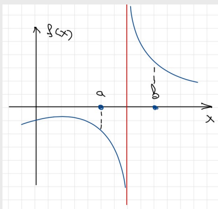
18

18

Bisectional method and singularity

If a nonlinear equation, such as $f(x) = 1/(x - d)$ which has a singularity at $x = d$, is bracketed between a and b, interval halving will locate the discontinuity, $x = d$.

A check on $|f(x)|$ as $x \to d$ would indicate that a discontinuity, not a root, is being found.



19

19

2.2 False position method

In the false position method, the nonlinear function $f(x)$ is assumed to be a linear function $g(x)$ in the interval (a, b), and the root of the linear function $g(x)$, $x = c$, is taken as the next approximation of the root of the nonlinear function $f(x)$, $x = c$.

The root of the linear function $g(x)$, that is, $x = c$, is not the root of the nonlinear function $f(x)$. It is a false position, which gives the method its name.

The method uses information about the function $f(x)$.

20

20

2.2 False position method - algorithm

The slope of the linear function $g'(x)$ is given by

$$g'(x) = \frac{f(b) - f(a)}{b - a} = \frac{f(b) - f(c)}{b - c}$$

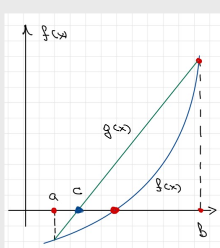
Setting $f(c) = 0$ and solving for c gives

$$c = b - f(b)\frac{b - a}{f(b) - f(a)}$$

then like for bisectional method

if $f(a)/f(c) < 0 \;\; a = a, b = c$

if $f(a)/f(c) > 0 \;\; a = c, b = b$



21

21

Example: C++

```
double false_p(double a, double b, double eps)
{
    double xl,x0,xr;

    if( f(a)*f(b) > 0.0) return 999;

    xl = a;
    xr = b;
    while (fabs(xr - xl) >= eps)
    {
      x0 = xr - f(xr)*(xr - xl)/(f(xr)-f(xl));
      if((f(xl) * f(x0)) <= 0.0 ) xr = x0;
      else xl = x0;
     }
       return x0;
}
```

22

22

Example for y = x - cos(x) on [0.0,4.0] for eps = 1.0e-6

```
 i    a       f(a)      b       f(b)      c       f(c)
 1  0.00000 -1.00000  4.00000  4.65364  0.70751 -0.05248
 2  0.70751 -0.05248  4.00000  4.65364  0.74422  0.00861
 3  0.70751 -0.05248  0.74422  0.00861  0.73905 -0.00006
 4  0.73905 -0.00006  0.74422  0.00861  0.73909 -0.00000
 5  0.73909 -0.00000  0.74422  0.00861  0.73909 -0.00000
 6  0.73909 -0.00000  0.74422  0.00861  0.73909 -0.00000
 7  0.73909 -0.00000  0.74422  0.00861  0.73909 -0.00000
 8  0.73909 -0.00000  0.74422  0.00861  0.73909  0.00000
iterations      root
     8       0.73909
```

for bisectional method it takes 22 iterations

The false position method generally converges more rapidly than the bisection method, *but it does not give a bound on the error of the solution*.

23

23

Part 3:

Open domain methods

24

Open domain methods

Open domain methods use information about the nonlinear function itself to refine the estimates of the root. Thus, they are considerably more efficient than bracketing ones.

However, such methods do not restrict the root to remain trapped in a closed interval. Consequently, they are not as robust as bracketing methods and can actually diverge.

Most popular open domain methods

1. Newton's method
2. The secant method
3. Muller's method

25

3.1 Newton’s method

Newton's method exploits the derivatives $f'(x)$ of the function $f(x)$ to accelerate convergence for solving $f(x) = 0$.

It always converges if the initial approximation is sufficiently close to the root, and it converges quadratically.

Its only disadvantage is that the derivative $f'(x)$ of the nonlinear function $f(x)$ must be evaluated.

26

Newton’s method – basics and algorithm

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + (x - x_0)^2 \frac{f''(x_0)}{2!} + \cdots$$

Suppose that x is the solution for $f(x) = 0$.

If we keep two first terms in Taylor series

$$f(x) = 0 = f(x_0) + (x - x_0)f'(x_0)$$

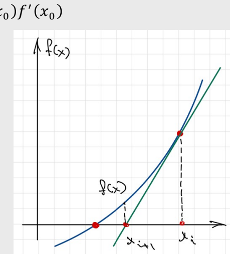
and then

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

We need $f(x)$ and $f'(x)$ to proceed

Each next iteration is

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$



27

Example: C++

```
double newton(void(*f)(double, double&, double&),double x, double eps,
int& flag)
{
    double fx, fpx, xc;
    int i, iter=1000;
    i = 0;
    do {
        i = i + 1;
        f(x,fx,fpx);
        xc = x - fx/fpx;
        x = xc;
        if(i >= iter) break;
        } while (fabs(fx) >= eps);
    flag = i;
    if (i == iter) flag = 0;
    return xc;
}
```

28

28

Example for y = x - cos(x) on [0.0,4.0] for eps = 1.0e-6

Initial point is 1.0

```
iterations      root
     4        0.73909
```

for bisectional method	22 iterations
for false position	8 iterations
Newton's method	4 iterations

Newton's method has excellent local convergence properties.

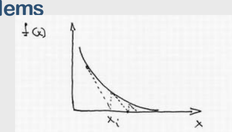
However, its global convergence properties can be very poor, due to the neglect of the higher-order terms in the Taylor series.

29

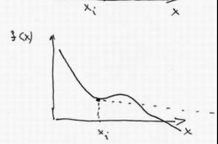
29

Newton's method – possible problems

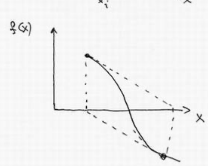
Very slow approach to the solution when $f'(x) \to 0$ around the root



difficulty with local minima, sending the next iteration value x_{k+1} far away



lack of convergence for asymmetric functions $f(a+x) = -f(a-x)$



30

30

Comments to Newton's method

Newton's method is an excellent method for polishing roots obtained by other methods which yield results polluted by round-off errors

Newton's method has several disadvantages.

1. Some functions are difficult to differentiate analytically, and some functions cannot be differentiated analytically at all.
2. If the derivative is small the next iteration may end up very far from the root

Practical comment:
In any program we must check the size of the step for the next iteration. If it is improbably large – then reject it (or switch to some other method)

31

31

3.2 Method of secants

The secant method is a variation of Newton's method when the evaluation of derivatives is difficult.

The nonlinear function $f(x)$ is approximated locally by the linear function $g(x)$, which is the secant to $f(x)$, and the root of $g(x)$ is taken as an improved approximation to the root of the nonlinear function $f(x)$.

32

32

3.2 Method of secants - algorithm

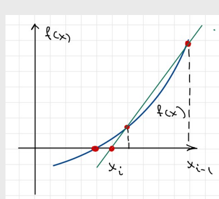
The derivative $f'(x)$ at point x_k can be approximated as

$$f'(x_k) = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

From the Newton's method

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}$$

One has to select two initial points to start



Note that the method of secant = the False position method

The only difference is about selecting two points to start the method

33

33

Example: C++

```
double secant (double(*f)(double), double x1,
               double x2, double eps, int& flag)
{
    double x3;
    int i, iter=1000;
    flag = 1;
    i = 0;
    while (fabs(x2 - x1) >= eps)
    {
     i = i + 1;
     x3 = x2 - (f(x2)*(x2-x1))/(f(x2)-f(x1));
     x1 = x2;
     x2 = x3;
     if(i >= iter) break;
     }
    if (i == iter) flag = 0;
    return x3;
}
```

34

34

Example for y = x - cos(x) on [0.0,4.0] for eps = 1.0e-6

Initial point is 1.0

```
iterations       root
      5        0.73909
```

for bisectional method	22 iterations
for false position	8 iterations
Newton's method	4 iterations
the secant method	5 iterations

Which method is more efficient?

Jeeves showed that if the effort required to evaluate $f(x)'$ is less than 43 percent of the effort required to evaluate $f(x)$, then Newton's method is more efficient. Otherwise, the secant method is more efficient.

35

35

3.2 Muller's and Brent's methods

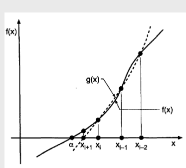
Muller's method is based on locally approximating the nonlinear function $f(x)$ by a quadratic function $g(x)$, and the root of the quadratic function $g(x)$ is taken as an improved approximation to the root of the nonlinear function $f(x)$.

Three initial approximations x_1, x_2, and x_3, (which are not required to bracket the root), are required to start the algorithm.

The only difference between Muller's method and the secant method is that $g(x)$ is a quadratic function in Muller's method and a linear function in the secant method.

Brent's Method is a hybrid method—it uses parts of solving techniques from other methods.

Many numerical libraries, e.g. MatLab, implement a version of Brent's method.



36

Summary for the open domain methods

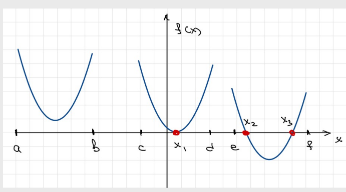
- All three methods converge rapidly in the vicinity of a root.
- When the derivative $f'(x)$ is difficult to determine or time consuming to evaluate, the secant method is more efficient.
- In extremely sensitive problems, all three methods may misbehave and require some bracketing technique.
- All three of the methods can find complex roots simply by using complex arithmetic.

37

37

Complications

- there are no roots at all
- there is one root, but the function does not change the sign,
  $$f(x) = x^2 - 2x + 1$$
- there are two or more roots on an interval $[a, b]$



What will happen if we apply the bisectional method here?

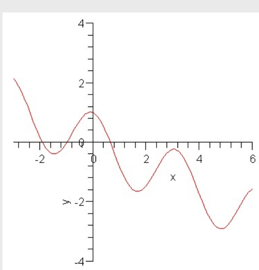
How about Newton's method?

38

38

Complications (cont.)

Many roots!



What root will you find with the bisectional method?

39

39

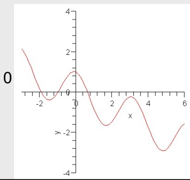
Multiple roots: "brute force" method

The brute force method is a good approach for dealing with multiple roots.

You split the original interval $[a, b]$ into smaller intervals with some step h applying some of the methods for single roots to each subinterval.

Choosing a step size

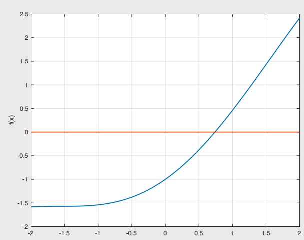
- If the step size is too large, we may miss multiple zeros.
- Choosing too small steps may result in time consuming calculations.
- A graphical analysis of the equation may help to decide for the most reasonable step for h.
- A good idea – evaluate roots for steps h and h/10 whether the number of roots stay the same.



40

Example 1:

$f(x) = x - \cos x$



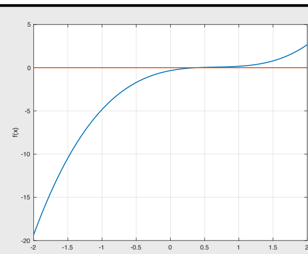
```
 Root(s) of f(x)
 Interval [-2.0, 2.0]
 Secant    x0 = 1.0
 Tolerance = 1.00e-08
                   root      iterations     f(root)
Bisectional       0.739085       29        -0.000000
False position    0.739085       11        -0.000000
Secant method     0.739085        6         0.000000
Matlab solution   0.739085                  0.000000

  Brute force roots
  number   root       f(root)
    1    0.739085    0.000000
```

41

41

Example 2:

$f(x) = x^3 - 2x^2 + 1.5x - 1/3$



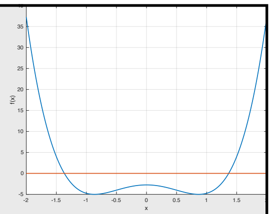
```
 Root(s) of f(x)
 Interval [-2.0, 2.0]
 Secant    x0 = 1.0
 Tolerance = 1.00e-08
                   root      iterations     f(root)
Bisectional       0.373462       29        -0.000000
False position    0.373462      347         0.000000
Secant method     0.373462       10        -0.000000
Matlab solution   0.373462                  0.000000

  Brute force roots
  number   root       f(root)
    1    0.373462    0.000000
```

42

42

Example 3:

$f(x) = 4x^4 - 6x^2 - 11/4$



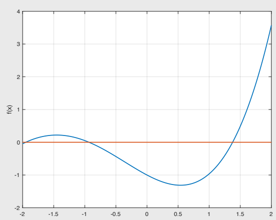
```
 Root(s) of f(x)
 Interval [-2.0, 2.0]
 Secant    x0 = 0.5
 Tolerance = 1.00e-08
                    root      iterations     f(root)
Bisectional:     No root found
False position:  No root found
Secant method     0.500012        18        -4.000046
Matlab solution   1.366760                   0.000000

  Brute force roots
  number   root       f(root)
    1    -1.366760    0.000000
    2     1.366760    0.000000
but for x0=1.0 Secant method gives 1.366760 after 11 iterations
```

43

43

Example 4:

$f(x) = \exp(x) - 2\sin(x) - 2$



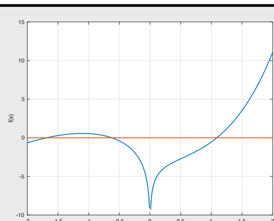
```
 Root(s) of f(x)
 Interval [-2.0, 2.0]
 Secant    x0 = 1.0
 Tolerance = 1.00e-08
                    root      iterations     f(root)
Bisectional       1.376879        29        -0.000000
False position   -1.950219         5         0.000000
Secant method     1.376879         8        -0.000000
Matlab solution   1.376879                   0.000000

  Brute force roots
  number   root       f(root)
    1    -1.950219   -0.000000
    2    -0.932760    0.000000
    3     1.376879    0.000000
```

44

44

Example 5:

$f(x) = \exp(x)\ln(x^2) - x\cos(x)$



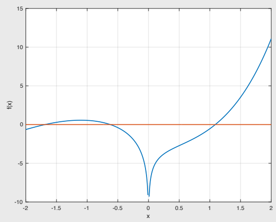
```
 Root(s) of f(x)
 Interval [-2.0, 2.0]
 Secant    x0 = 1.0
 Tolerance = 1.00e-08
                    root      iterations     f(root)
Bisectional       1.088682        29        -0.000000
False position   -1.685852        20        -0.000000
Secant method     1.088682         7        -0.000000
Matlab solution   1.088682                  -0.000000

  Brute force roots
  number   root       f(root)
    1    -1.685852    0.000000
    2    -0.623612    0.000000
    3     1.088682    0.000000
```

45

45

Example 5: change x0

$f(x) = \exp(x)\ln(x^2) - x\cos(x)$



```
 Root(s) of f(x)
 Interval [-2.0, 2.0]
 Secant    x0 =-1.0
 Tolerance = 1.00e-08
                    root       iterations      f(root)
Bisectional       1.088682        29         -0.000000
False position   -1.685852        20         -0.000000
Secant method    -1.685852        13          0.000000
Matlab solution  -0.623612                    0.000000

  Brute force roots
  number    root       f(root)
    1    -1.685852    0.000000
    2    -0.623612    0.000000
    3     1.088682    0.000000
```

46

46

Part 4:

Roots of polynomials

$$a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = 0$$

47

Ideas

The fundamental theorem of algebra states that a nth-degree polynomial has exactly n zeros, or roots.

The roots may be real or complex. If the coefficients are all real, complex roots always occur in conjugate pairs. The roots may be single (i.e., simple) or repeated (i.e., multiple).

Descartes' rule of signs, which applies to polynomials having real coefficients, states that the number of positive roots of $P_n(x)$ is equal to the number of sign changes in the nonzero coefficients of $P_n(x)$ or is smaller by an even integer.

The number of negative roots is found in a similar manner by considering $P_n(-x)$.

48

48

The bracketing methods for $P_n(x)$

The bracketing methods (bisection and false position), cannot be used to find repeated roots with an even multiplicity, since the nonlinear function $f(x)$ does not change sign at such roots.

Repeated roots with an odd multiplicity can be bracketed by monitoring the sign of $f(x)$, but even in this case the open methods are more efficient.

49

49

The open methods for $P_n(x)$

The open can be used to find the roots of polynomials: Newton's method, the secant method, and Muller's method.

These three methods also can be used for finding the complex roots of polynomials, provided that complex arithmetic is used, and reasonably good complex initial approximations are specified.

There are various modifications of Newton's method for polynomials

Other methods for polynomials: Bairstow's method, Laguerre's method, Eigenvalue method.

Matlab has a function **roots** to return roots of polynomials.

50

50

Part 5:

Nonlinear systems of equations

51

Systems of non-linear equations

$$\begin{cases} f(x,y) = 0 \\ g(x,y) = 0 \end{cases}$$

"There are no good, general methods for solving systems of more than one nonlinear equation"

Numerical recipes in C by W. H Press et al.

- Bracketing methods are not readily extendable to systems of nonlinear equations.
- Newton's method, however, can be extended to solve systems of nonlinear equations. Quite often you need a good initial guess.

52

52

Newton's method $\begin{cases} f(x,y) = 0 \\ g(x,y) = 0 \end{cases}$

Find x^* and y^* such that $f(x^*, y^*) \approx 0,\ g(x^*, y^*) \approx 0$

Using Taylor series about (x, y)

$$f(x^*, y^*) = f(x,y) + f_x'(x,y)(x^* - x) + f_y'(x,y)(y^* - y) + \cdots$$

$$g(x^*, y^*) = g(x,y) + g_x'(x,y)(x^* - x) + g_y'(x,y)(y^* - y) + \cdots$$

keeping only first-order terms and setting $f(x^*, y^*) \approx 0,\ g(x^*, y^*) \approx 0$

one has a system of linear equations for x^* and y^*. Solving it gives

$$x^* = x + \frac{f_y'(x,y)g(x,y) - f(x,y)g_y'(x,y)}{f_x'(x,y)g_y'(x,y) - f_y'(x,y)g_x'(x,y)}$$

$$y^* = y + \frac{f\ (x,y)g_x'(x,y) - f_x'(x,y)g\ (x,y)}{f_x'(x,y)g_y'(x,y) - f_y'(x,y)g_x'(x,y)}$$

53

53

Example: C++

```
void newton2(double& x1, double& y1, double eps, int& i)
{
   double     f1, g1, fx, fy, gx, gy;
   double     del, x2, y2, dx, dy;
   int iter = 99;
   i = 0;
   do {
       i = i + 1;
       fg(x1, y1, f1, g1, fx, fy, gx, gy);
       del = fx*gy - fy*gx;
         dx=(fy*g1-f1*gy)/del;
         dy=(f1*gx-fx*g1)/del;
         x2=x1+dx;
         y2=y1+dy;
       x1=x2;
         y1=y2;
       if(i >= iter) break;
   } while (fabs(dx) >= eps && fabs(dy) >=eps);
   i = i+1;
}
```

54

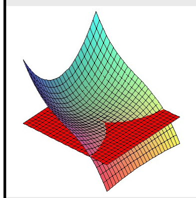
54

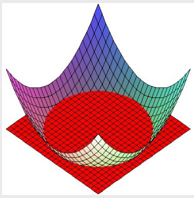
Example

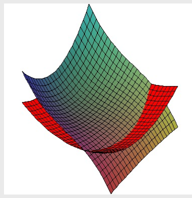
$$\begin{cases} y^2(1-x) = x^3 \\ x^2 + y^2 = 1 \end{cases}$$

$f(x,y) = y^2(1-x) - x^3$ $\quad g(x,y) = x^2 + y^2 - 1$ $\quad f(x,y) = y^2(1-x) - x^3$

$h(x,y) = 0$ $\quad h(x,y) = 0$ $\quad g(x,y) = x^2 + y^2 - 1$







55

55

Example

$$\begin{cases} y^2(1-x) = x^3 \\ x^2 + y^2 = 1 \end{cases}$$

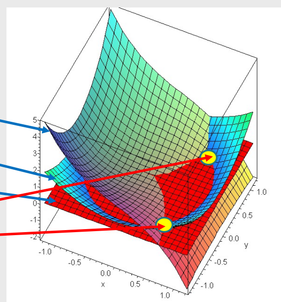
plots

$f(x,y) = y^2(1-x) - x^3$

$g(x,y) = x^2 + y^2 - 1$

$h(x,y) = 0$

solutions



56

Example with various initial points

Newtons method for two coupled nonlinear equations

i	x	y	f	g	dx	dy
1	1.00000	1.00000	-1.00000	1.00000	-0.25000	-0.25000
2	0.75000	0.75000	-0.28125	0.12500	-0.11905	0.03571
3	0.63095	0.78571	-0.02335	0.01545	-0.01276	0.00041
4	0.61819	0.78613	-0.00030	0.00016	-0.00016	0.00002
5	0.61803	0.78615	-0.00000	0.00000	-0.00000	0.00000
6	0.61803	**0.78615**	-0.00000	0.00000		

Newtons method for two coupled nonlinear equations

i	x	y	f	g	dx	dy
1	1.00000	-1.00000	-1.00000	1.00000	-0.25000	0.25000
2	0.75000	-0.75000	-0.28125	0.12500	-0.11905	-0.03571
3	0.63095	-0.78571	-0.02335	0.01545	-0.01276	-0.00041
4	0.61819	-0.78613	-0.00030	0.00016	-0.00016	-0.00002
5	0.61803	-0.78615	-0.00000	0.00000	-0.00000	-0.00000
6	0.61803	**-0.78615**	-0.00000	0.00000		

57

57

Part 6:

Summary

58

Root-finding algorithms should contain:

1. An upper limit on the number of iterations.
2. If the method uses the derivative $f'(x)$, it should be monitored to ensure that it does not approach zero.
3. A convergence test for the magnitude of the solution, $|x_{i+1} - x_i| \le \varepsilon$, or/and the magnitude of the nonlinear function, $|f(x_{i+1})| \le \varepsilon$, must be included.
4. When convergence is indicated, the final root estimate should be inserted into the nonlinear function $f(x)$ to guarantee that $f(x) = 0$ within the desired tolerance.

59

59

Summary 1:

1. Bisection and false position methods converge very slowly but are certain to converge because the root lies in a closed domain.
2. Newton's method and the secant method are both effective methods for solving nonlinear equations. Both methods generally require reasonable initial approximations.
3. Polynomials can be solved by any of the methods for solving nonlinear equations.
   However, the special features of polynomials should be taken into account.

60

60

Summary 2:

3. Multiple roots can be evaluated using Newton's basic method or its variations, or brute force method
4. Complex roots can be evaluated by Newton's method or the secant method by using complex arithmetic.
5. Solving systems of nonlinear equations is a difficult task.
   For systems of nonlinear equations which have analytical partial derivatives, Newton's method can be used.
   Otherwise, multidimensional minimization techniques may be preferred.
   No single approach has proven to be the most effective.
   Solving systems of nonlinear equations remains a difficult problem.

61

61

Summary 3:

6. Good initial approximations are extremely important.
7. For smoothly varying functions, most algorithms will always converge if the initial approximation is close enough.
8. Many, if not most, problems in engineering and science are well behaved and straightforward.
9. When a problem is to be solved only once or a few times, the efficiency of the method is not of major concern. However, when a problem is to be solved many times, efficiency of the method is of major concern.
10. If a nonlinear equation has complex roots, that must be anticipated when choosing a method.
11. Analyst's time versus computer time must be considered when selecting a method.

62

62

Pitfalls of Root Finding Methods

1. Lack of a good initial approximation
2. Convergence to the wrong root
3. Closely spaced roots
4. Multiple roots
5. Inflection points
6. Complex roots
7. Ill-conditioning of the nonlinear equation
8. Slow convergence

63

63

Other Methods of Root Finding

Brent's method uses a superlinear method (i.e., inverse quadratic interpolation) and monitors its behavior to ensure that it is behaving properly.

For finding the roots of polynomials: Graeff's root squaring method, the Lehmer-Schur method, and the QD (quotient-difference) method. Two of the more important additional methods for polynomials are Laguerre's method and the Jenkins-Traub method

64

64

Packages for non-linear equations

Numerous libraries and software packages are available for solving nonlinear equations. Many workstations and mainframe computers have such libraries attached to their operating systems

Many commercial software packages contain nonlinear equation solvers (e.g. Matlab, Mathematica, Maple, Mathcad).

More sophisticated packages can be found in IMSL, NAG.

The book Numerical Recipes (Press et al., many editions) contains numerous subroutines for solving nonlinear equations.

65

65

Final thoughts

1. Choosing right computational method for finding roots is a difficult skill for beginners.
2. A method that was efficient for one equation may fail miserably for another
3. Any method should be used intelligently!

66

66